COMP2396B - Assignment 7

Due: **24 May, 2020 23:59**

Introduction

This assignment tests your skills on writing **networking and multi-threading** program in Java.

You are required to write a **peer-to-peer** (P2P) images sharing system. A **hybrid P2P structure** is used in this assignment, where a server is used as the source of **images** as well as that of the **list of peers** available. In this assignment, you need to integrate the program written in **Assignment 4 (authentication module)** and **24-hours Programming Tasks** to complete this assignment.
You need to write **two main programs**. `ImageServer.java` is the server program, and `ImagePeer.java` is the client (peer) program. Notice that a server will also act as a peer to share images.

Part 0. User.txt generalization

Use and extend code in a4 to generate User.txt. We will use the function **2.Add user record** in assignment 4 to store the following user information.

Username: mqpeng	Password: Comp2396B
Username: cjli	Password: 2396BComp
Username: cbchan	Password: HelloWorld0

We export the username and hashed password to the User.txt. Since there is no File I/O function in assignment 4, you need to add the I/O function. The sample format of User.txt is shown here:

username:mqpeng;hashPassword:9d8814d33e6ebb43f7864881c37819fffcd87
username:cjli;hashPassword:4bae614bd712d6b371b32465e81579ec15e43eb
username:cbchan;hashPassword:ef4faa8b853d3e28d78867ff15afe7fa3e9cc28

The value of hashPassword field is hashed password [eg. 9d8814d33e6ebb43f7864881c37819fffcd87] generated from the plain text password [Comp2396B]. The result of the hashPassword value is no required to be exactly the same to the shown sample value. Since you may choose different hash function, the result may be different.

The code in this part will not be graded. So, for part 0, please submit the User.txt only.

Part 1. Server interface

When `ImageServer.class` is executed, a file chooser should be presented to ask for an image file. (Figure 1)

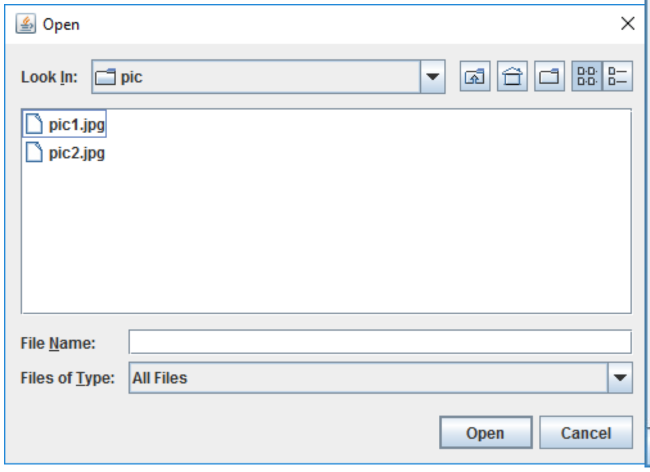


Figure 1: File chooser

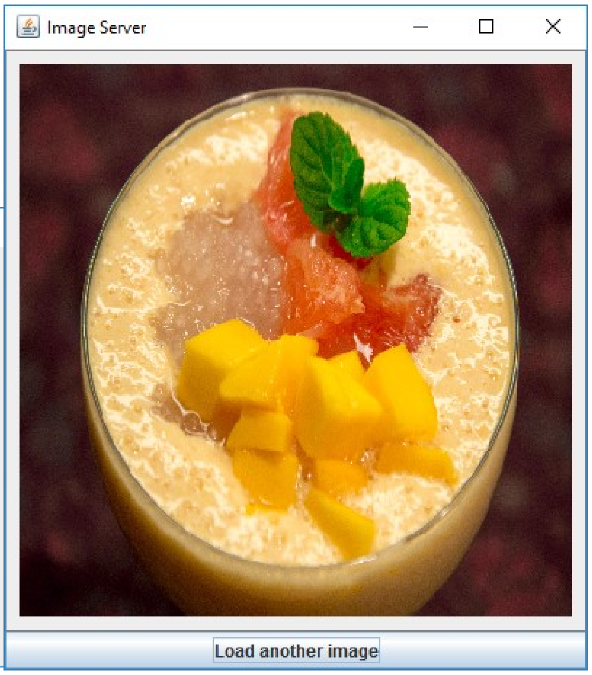


Figure 2: Server GUI

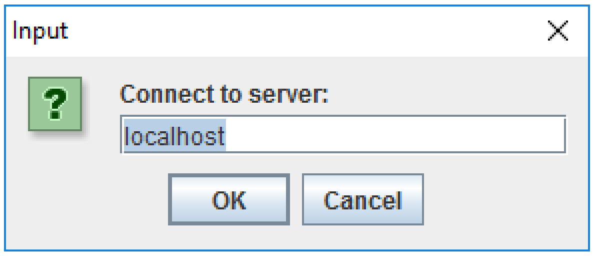
The image is then loaded and displayed as shown in Figure 2. If the image file fails to load, the program terminates. Otherwise, the server should listen to the port **9000**, and load the **User.txt** which contains the user information into the program and waiting for user to login. The user information in User.txt could be used to login and add into the P2P images sharing system.

The image should be **resized and scaled** to fit into a canvas of 700*700 pixels in size. The button "Load another image" allows the user to change the current image. If the new image fails to load, the old image is retained.

Part 2. Peer program and authentication

When `ImagePeer.java` is executed, it should ask for the server's IP address to be connected to. It then asks the peer user to login. (Figures 3, 4 and 5). For the user authentication, you are required to use the **authentication module implemented in Assignment 4** as the interface.

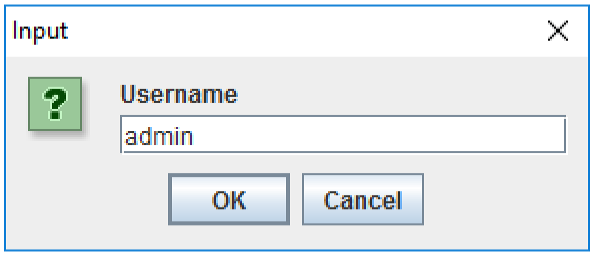
The client should use the **hash interface** to hash the password before sending to the server and the server need to verify the user information before starts the main GUI. Hence, the server should also return the message if the user cannot login. (e.g. Account is locked)


Input

Connect to server:

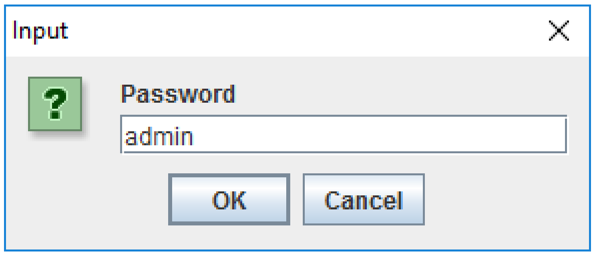
localhost

Figure 3: Username input


Input

Username

admin

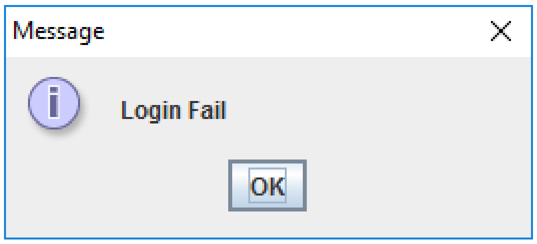

Input

Password

admin

Figure 4: Username input **Figure 5: Password input**

If the login is failed or locked, it should show an alert. (Figure 6)


Message

Login Fail

Figure 6: Login alert

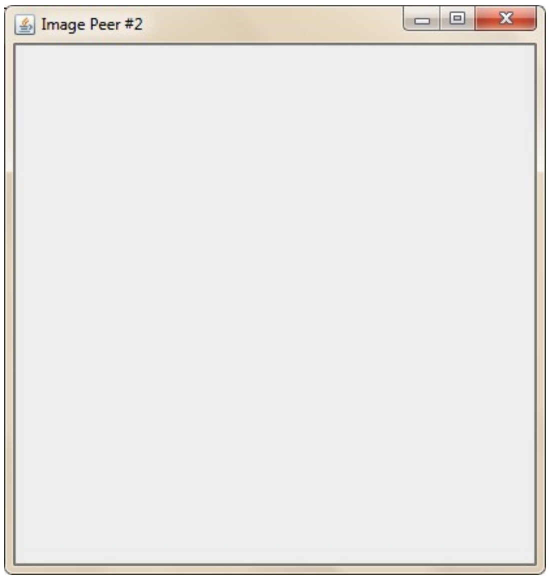


Figure 7 Peer GUI

If the login is successful, an empty canvas should then be presented as shown in Figure 7. Like the server, the canvas size in the peer should also be **700*700 pixels**.

Immediately after the GUI is shown, the peer should connect to the server and start to download the image from the server and other peers. Program should be terminated if it fails to connect to the server. (See details below)

<u>Part 3: Peer initialization</u>

The server should listen to port **9000** after initialization. The server has to maintain the following:

1. The image, separated into blocks of **70*70 pixels** in size
2. A list of **active peers** (the peer IP address and port number)

When a peer program is started, it should connect to the server at port **9000**. The server should then update the list of active peers, and each peer should collect the list of current active peers from the server. This connection can be closed afterwards.

<u>Part 4: P2P operation</u>

Each peer should perform the following:

1. Try to **download blocks** of image from the list of peers in **simultaneously**.
2. **Accept** connection from other peers and **send out blocks** of images other peers needed.

Notice that the server itself is a special peer that is the source of the image to be distributed. Details about the suggested P2P operation will be introduced in the tutorial.

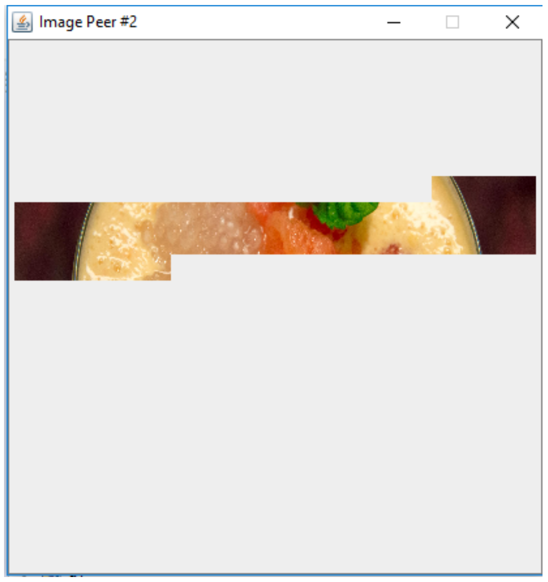
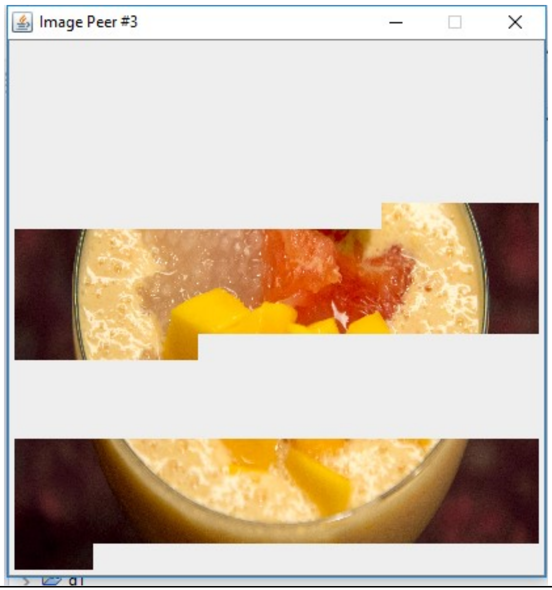
Part 5: Update image

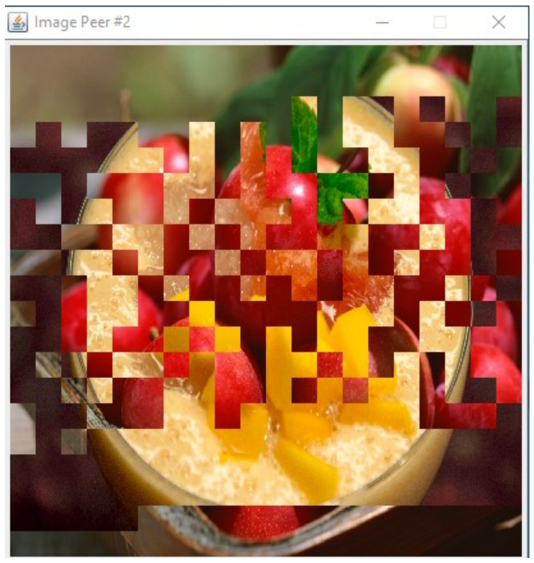
The image should be updated if:

1. If the user **switches** the image from the server program, all peers will change its image **immediately**.
2. The user **drags and drops** the image block using the server program GUI. For such case, the client only required to download the **swapped image blocks**.

Example execution

Execution result depends on how the P2P operation is implemented. Here is an example of the execution behavior. (This is the process of downloading images in slow-motion display, you do not need to show this part of the implementation result.)

When the first peer starts, it will get the peer list [server] first. Then, it starts downloading blocks of image from the only peer (the server) available.	
When the second peer starts, it will get the peer list [server, peer1] first. Then, it starts downloading from the server peer (the bottom part) and the first peer (the middle part). The peer will ask other peers for different part of image. You can decide your own way about how the distribution works.	

When the user switches the image in the server, the peers which choose to update the image will download new blocks from the server and other peers.	

Marking

- **85% marks** are given to the **functionality**.
  - ➢ You may add additional classes, instant variables and methods to the project.
  - ➢ You may need to include and modify the classes implemented in assignment 4 for the authentication function.
  - ➢ You will get part of the marks if you implemented some of the features.
  - ➢ A program that can run normally without throwing exceptions during runtime gets higher marks.
  - ➢ Program with compile error will get 0 mark.
- **15% marks** are given to your **JavaDoc**. A complete JavaDoc includes documentation of every classes, member fields and methods that are not private. JavaDoc for the main method may be omitted.

Submission

Please submit all source files (all *.java file) in a single compressed file (in .zipor .7z) to Moodle. Please also include your generated 'User.txt' file and no submission for the generation code in part 0. **Late submission is not allowed**. **Program with compile error will get 0 mark. (Please include any code that help compile)** You are welcome to submit the Readme.txt which introduces how to run your code.

Do not submit .*class* file.

Plagiarism

Do not attempt plagiarism. We will check your program with software that checks program structure. Both the source and the copying work will be penalized.

-- END --