Assignment 3

Deadline: Sunday, October 18, 23:59

Instruction

This assignment is backed by Chapter 11. You have the maximum control of the way you program (subjected to requirements, if mentioned), but:

- Your code should be well-written in terms of the design (careful considerations and bug-freeness) and efficiency (few duplicates).
- The program must be adequately commented and follow a coding style. See the "Introductory video" on Canvas to find references for basic rules of writing comments and programming style.

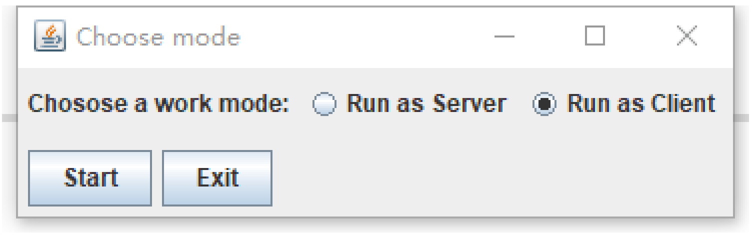
Question 1

Android applications (app) are mainly written in Java. Roughly speaking, in an android app, each screen is an "activity", and the switch of screens is implemented as a switching of activities. We are now going to implement a simplified version of this using the standard Java language (there is probably no need to read articles on Android programming for this question).

We are going to write a **fake** chatting app. Here we use the word "fake", which means that we are not going to actually build a chatting app with all the typical functionality of that type of app, which usually operates through a distributed system and is capable of chatting between multiple clients, etc. A functional chatting app would be unfortunately cumbersome to be implemented within 2 weeks. However, in this assignment we are going to implement an important component of chatting apps. Your task is to implement an interface somewhat typical of this type of apps. This app has a server part and a client part; and has 4 screens/activities, each corresponding to a class. Next, we explain how that works. Note that many things (buttons, etc.) are self-explanatory, so we skip the explanation of them. **In addition, please read until the end to see the requirements.**

Activity 1 - `MainActivity`

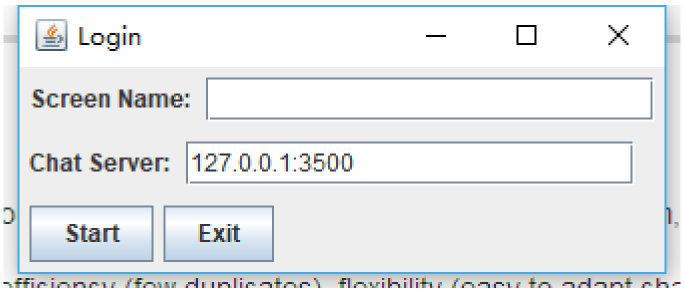
When the program starts, the `MainActivity` class/activity fires up to produce the following window.



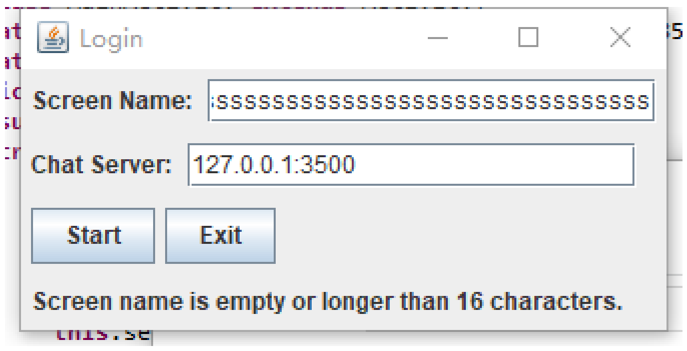
The user chooses a work mode from Client or Server.

Activity 2 - `StartClientActivity`

If the user chooses the "Run as Client" mode, another class/activity, `StartClientActivity`, runs, and the following window shows up.



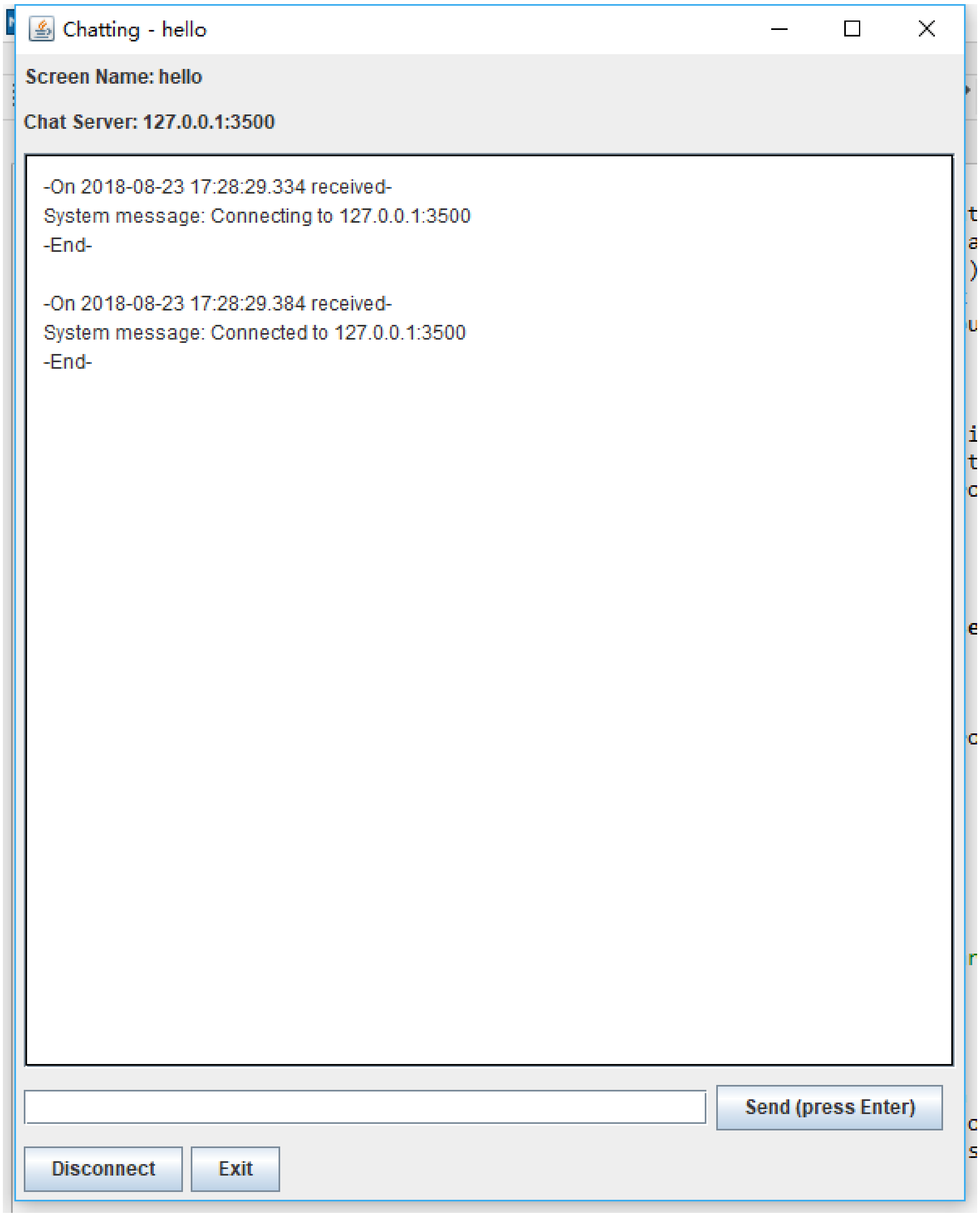
And, the `MainActivity` activity disappears. Here the user has to fill in a screen name and a chat server. The "Chat Server" field should be pre-filled with "127.0.0.1:3500". In addition, the screen name should not be empty nor longer than 16 characters. If that is the case, shown an additional message when the user clicks "Start", as the following.



Screen name is empty or longer than 16 characters.

Activity 3 - `ClientActivity`

If everything is fine and the user clicks start, the `ClientActivity` class/activity starts, and `StartClientActivity` gone, with the following window.



Screen Name: hello

Chat Server: 127.0.0.1:3500

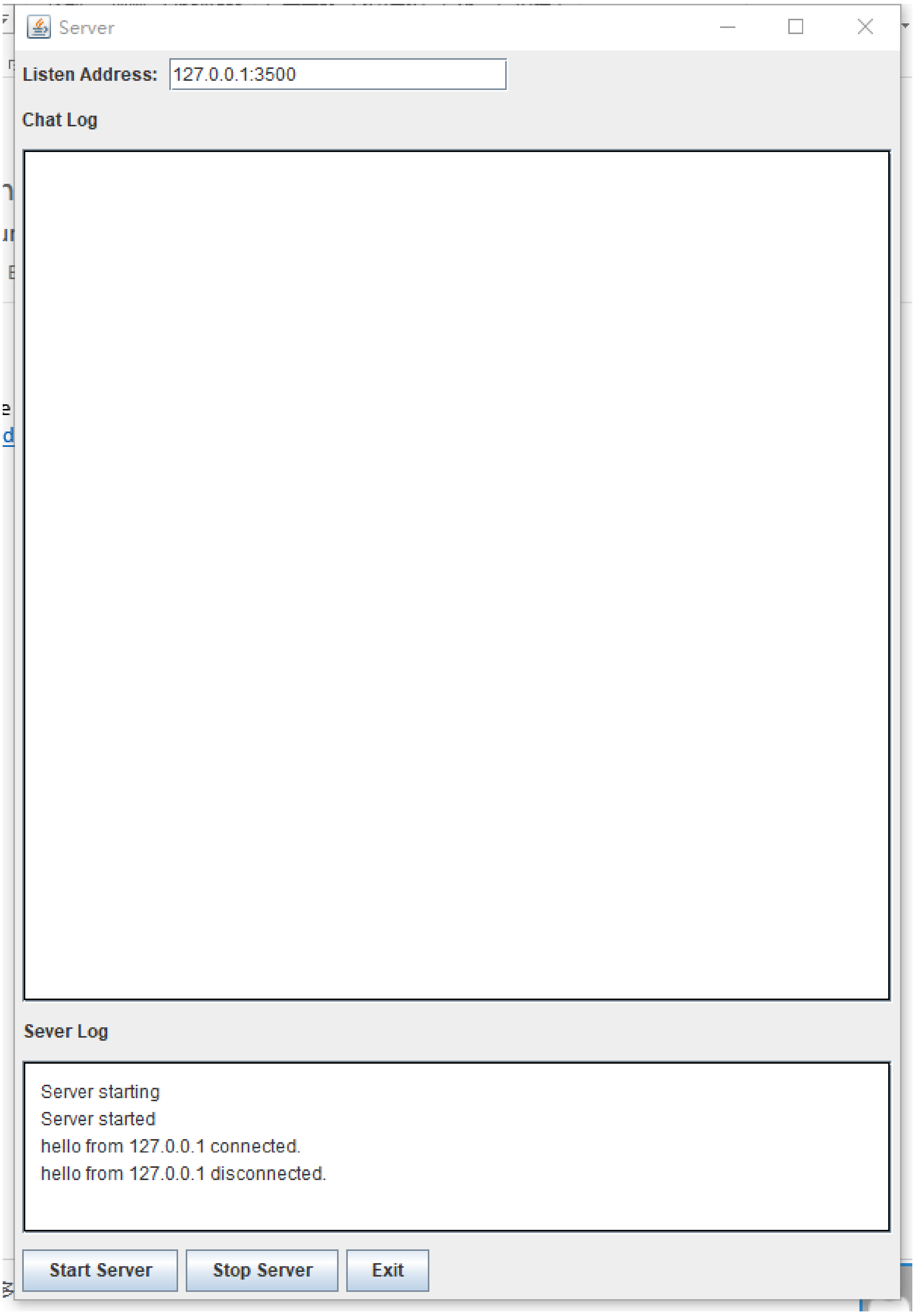
```
-On 2018-08-23 17:28:29.334 received-
System message: Connecting to 127.0.0.1:3500
-End-

-On 2018-08-23 17:28:29.384 received-
System message: Connected to 127.0.0.1:3500
-End-
```

Here, you should display the chosen screen name and the chat server as above, and the big text area is the chat log and is an instance of `JTextArea`. The smaller text field below is the chat box, which is where the user types the message. When the user clicks "Send (press Enter)", just ("fake-ly") put the message, if non-empty, up into the chat log, in a format same as the two pre-set messages. However, the two pre-set messages in the chat log are themselves optional. The "Disconnect" button does nothing. Here, when the user press Enter in the chat box (instead of clicking that "Send (press Enter)" button), the message should also be put in the chat log.

Activity 4 - `ServerActivity`

If the user choose "Run as Server" in `MainActivity`, `MainActivity` disappears but the `ServerActivity` class/activity pops up, as the following.



```
Server starting
Server started
hello from 127.0.0.1 connected.
hello from 127.0.0.1 disconnected.
```

This window is completely fake (that is, is not really functional), except that the "Listen Address" should be pre-filled with "127.0.0.1:3500", and that the "Exit" button should work. The messages below "Server Log" are optional.

Requirements

Here are some requirements on how you should write this fake app. The idea is that there should be an abstract class `Activity` that serves as a generic window manager. The 4 classes, or actual activities, mentioned above extends `Activity`. `Activity` represents an abstract/generic activity but the 4 classes above are specializations. Think about `Activity` to be the template and the 4 actual activities to be realizations of that template.

To be more specific, the abstract activity should be well-separated from actual activities in the following ways.

- `Activity` should only handle UI components placements, etc. That is, the 4 classes above do not directly create `JPanel`, `JLabel`, `JButton`, etc.; but should ask `Activity` to do so, telling it how the components should look like. As an example, the `Activity` class may have a method

```
public int setButton(..., String title, ...)
```

that is called in `MainActivity` as

```
this.setButton(...,"Exit",...);
```

which creates the "Exit" button on the window.

- On the other hand, `Activity` should not handle any operations, nor store information, in an activity-specific manner. As examples, `Activity` class should not have anything like

```
someTextField.setText("127.0.0.1:3500");
```

or any field like

```
private String screenName;
```

- The assignments of listeners should be done through `Activity`, but the actual listeners should be provided by the corresponding activity (i.e., not defined within `Activity`).

Question 2

In this assignment we want to create a computer program that allows two people to play a **Tic Tac Toc** with each other. You probably already know how to play Tic-Tac-Toe. The rules are simple:

- The game is played on an 3 x 3 grid.
- Two players play, one can place X's on the board, the other O's. Players take turns putting their marks in empty squares.
- The player who obtains 3 of his/her marks in a row (vertical, horizontal, or diagonal) is the winner.
- When all 9 squares are full, the game is over. If no player has 3 marks in a row, the game ends in a tie.

The computer's job is to keep track of the moves that the players make, inform them if anyone has won, and prevent them from making incorrect moves. It should first ask player "one" to make a move, asking for the row and column of where the "X" is to be placed. Then it should do the same for player "two", placing an "O" in the appropriate row and column. It should alternately ask the players, checking after each move if a player has won (or if the game is over because there is no more room let to place a piece).

You are encouraged to use your creativity! However, your program should have the following features:

- The game needs to tell whose turn it is.
- The game needs a reset button.
- The game needs to tell who won (or if the game ended up in a tie).