AP AX

A Multiplayer Game Using Sockets

Introduction: To develop a distributed system that structures and supports play within a multiplayer game.

Your task is to come up with a system based on the techniques and examples in the course, that implements a game that can be played by at least two people at the same time. Your system should involve one Java program that acts as a game server, and one that acts as a Swing-based game client. Each client should communicate with the server, using sockets. Clients should not directly communicate each other, but communicate via the server. You should design the application level protocol for the system, setting out the patterns of communication between client and server, which you can then use to implement and test your programs. The server should start up a new thread each time a client connects. Each thread should be responsible for serving one client. Prevent any potential race conditions and deadlocks by using facilities such as synchronised methods and locks.

Each player should be able to connect with at least one other via the server, and play a game that involves multiple turns or rounds. Allow game play to be ongoing, i.e. allow the game play to start again once one game has been played. The overall system will manage players' interaction, by implementing the application level protocol, and enforcing the rules of the game. Note that marks will be given for the quality and richness of the system design that demonstrates your understanding of distributed systems and concurrency (see 'How you will be assessed', below). Therefore, please focus more on making technically sound infrastructure, than creating complex graphical interfaces.

You have two options for the game that you implement. **Choose one** for your submission.

The first (and simpler) option is the board game *Draughts*, with rules as per the Rules section here:
https://en.wikipedia.org/wiki/English_draughts
but with none of the variations specified in the Rule Variation section of that same page. Note that the number of players is always 2.

The second (and more complex) option is the card game *Twenty-One*, with rules as described in the fourth section here, i.e. not *Vingt-Un,* or *Siebzehn und Vier,* or other descendant games:
https://en.wikipedia.org/wiki/Twenty-One_(card_game)
Note that the number of players has a minimum of 2 but can be higher; it may even change over time within a game session. Also, there are *stakes*, which you should implement as simple in-game tokens, i.e. please do not connect this to real money!

What you should submit

- A design document, as a PDF file. This should have an explanation of the basic aims, rules and mechanisms of the game, and (at a high level) how they are implemented—including the application level protocol for the game. It should also have a description of the classes and class hierarchies that you have designed and built, in the form of a UML class diagram or some text. The description should be clear enough that someone else could easily modify or extend your classes. If using UML, feel free to annotate it. I will not be marking the *correctness* of the use of UML, just the clarity.
- Your code. Just *.java* files please. If you're using Eclipse or another IDE, remove the package line from the top, so that I can compile from the command line. Your code should include a main for the server (in a file *Server.java*) and a main for the client (in a file *Client.java*). These are what I will run to see your system in action.

When I unzip your code, I will type: *javac *.java*, then *java Server,* and then (in another terminal or terminals) *java Client*. So, **no packages.**

All of these files (the .java files and the .pdf) should be zipped up and submitted via Moodle as a **zip**. Not a **7z** or a **rar** or whatever else. I will deduct 1 mark if these basic requirements for submission and compilation are not met.

How you will be assessed

The mark awarded will be out of 30 (as it's 30% of the course). Of this, 10 will be for the design, and 20 for the code. For the design, marks will be awarded for:

- Clarity
- Demonstrating the practical understanding of distributed systems and concurrency
- Sensibly designed application protocol
- Sensibly designed classes (e.g. objects that can be re-used, lack of code being repeated, correct use of abstract classes / interfaces as appropriate, etc.)

For the code, marks will be awarded if:

- It fulfills the requirements as described above (see note below)
- It is neat, easy to follow and well commented
- It conforms to Java naming conventions
- Correct use of object declarations (e.g. static and final)

If you are unable to create something that fulfills all of the requirements, don't panic. **There are plenty of marks available if you don't get everything working, even with the first game option (Draughts).** Try and ensure you do submit something that will compile and will run, even if it is not fully functional. Feel free to add comments to your code describing functionality that you weren't able to implement.

Deadline

4.30pm, Friday 27th March 2020.

Finally…

A few hints and comments:

- Examine the lecture, lab and Horstmann examples, and understand how they work.
- Take time to design the application protocol, and make sure that the client and server are always consistent with it (i.e. if the protocol changes, then it's likely that both programs will have to change).
- I realise that you are once again doing a game, and you may be tired of them. I apologise, but they are a useful means to explore a great variety of technical and infrastructural issues, since requirements can be coherent yet compact.
- **Remember**: this is worth 30% of the course. Failure to make any effort will mean you have attempted less than 75%, and will not be eligible for credit.