CS380 — Project 7

February 22, 2016

Due: Friday, March 4, 2016 (60 points)

Preparing the Project

1. Go to `https://codebank.xyz` and you should see a project in the CS380 group named `CS380-P7`. Fork this project for your own user.
2. Next, clone the repository so you have a local copy:

   ```
   $ git clone https://codebank.xyz/username/CS380-P7.git
   ```

3. The repository should have several Java classes already that act as the message objects we pass back and forth.

Description

In this project, we'll be using Java's `Serializable` interface to serialize application data and send it across a network. Remember that Java's `Socket` is an implementation of TCP. We've been using TCP sockets, giving us a reliable byte stream between applications. Once we serialize application data (coverting it to byte strings), we can send them across the byte streams we already have.

You will be creating a client program for playing Tic-tac-toe across a network. You will connect to a server and send serialized commands and messages to the server, then receive serialized responses back. Your client can handle the user interaction and display however you like, as long as it's clear how to interact with the program and play the game. You can create either a text-based or GUI client. We are, in a sense, doing event-driven programming, so it may help to learn about that paradigm if you haven't used it before.

I have provided the set of classes corresponding to the types of messages with which we can communicate. They have been designed as follows:

Class	Description
`Message`	Base class for all messages. This lets us assign a type to each message so we can see what type of message it is before attempting to cast it to a more specific type.
`ConnectMessage`	Send this immediately when connecting to provide a user name, similar to the original chat client you made.
`ErrorMessage`	When something goes wrong, I will send an instance of this class for you to check and see what the problem was.
`BoardMessage`	During play, I will send these messages to indicate the latest board configuration for the game.
`MoveMessage`	When you make a move, you will send one of these messages indicating where you are attempting to move.
`CommandMessage`	There are a few commands that you can give the server using this class. You can tell the server to start a new game, that you are exiting the program, or you can tell the server that you surrender if you are currently in a game.

Do not attempt to modify any of the provided classes. It may make the serialization incompatible and any changes you make will not be considered when grading your submission. If you try to submit modified versions of any of the supplied classes, your changes will be ignored and the grader will test it with our own copies, so it will probably break your program!

For a proper submission, your program needs to be able to complete one game against the server. If you want, you can have it start a new game afterwards. Other commands are optional.

Your program must first send a `ConnectMessage` identifying yourself. Next, you should send a `CommandMessage` to start a new game with the server. You are always the first player and the server will play second. After starting a new game, the server will respond with a `BoardMessage` showing the starting board configuration. Next, send a `MoveMessage` indicating where you are making your move. The server will make its move and reply with another `BoardMessage`. This will continue until the game ends. Note that your program doesn't really need to be able to play Tic-tac-toe at all, you simply need to pass the messages to the server and display the results. The server will handle the gameplay.

If a problem occurs, the server will respond with an `ErrorMessage`. Like the chat program from earlier in the quarter, it will probably be useful to use a separate listening thread for this project.

The `BoardMessage` and `MoveMessage` classes use `bytes` to represent board locations and values. For `MoveMessage`, think of `row` and `col` as array indices so you should start counting from zero. For `BoardMessage`, the board is represented by a 3×3 byte array, where the values stored in the array will be 0 for an empty square, 1 for a square held by player 1 (typically X), and 2 for a square held by player 2 (typically O). A `BoardMessage` also contains information such as the status of the game and the turn number.

The server will be running on the same host as previous projects: `cs380.codebank.xyz` on port 38007.

Submission

Your project should have a main class named `TicTacToeClient` in a file named `TicTacToeClient.java`. You can have other files or classes, but it should successfully compile and run by simplying using:

```
$ javac TicTacToeClient.java
$ java TicTacToeClient
```

You may push changes as many times as desired before the deadline.