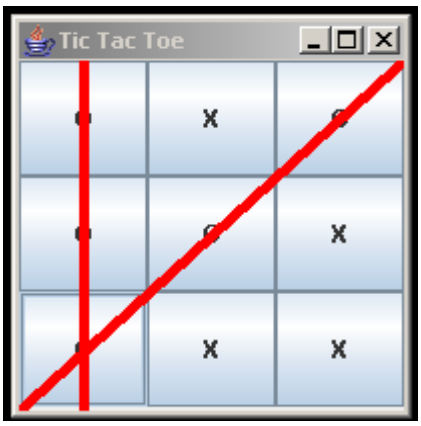
COMP 1006/1406
Assignment 4 – TicTacToe (Part 2)

Due: Friday, August 11th 2006, before 11h55 pm

In this assignment, you will practice programming networked applications. We will allow the TicTacToe game we have made in assignment 3 to work on two different computers at the same time. This will be done using the Socket interface of the Java API. You will also practice overloading the paint() method of a Java component.

Part A (A Nicer GUI): [20 pts]



That part is a warm-up. The goal is to paint a red line over every line which makes a player winning. The line has to be red and it has to be 5 pixels wide. In the TicTacToePanel class, you have to:

- Create a paintLine(Graphics g) method which does the following:
  - If the game is not finished, the method just returns.
  - If the game is draw, the method just returns.
  - If there is a winner, then over each row which allowed that player to win, draw a 5 pixels wide red line. Look at the javadoc in order to know how to change the color of a Graphics object. To change the width, you will have to cast your Graphics object into a Graphics2D object and change its Stroke.

Part B (Streaming the Game): [15 pts]

The high-level description of the strategy we will use in order to play TicTacToe online is the following:

- We will create a new sub-class of the Player class called RemotePlayer.
- A RemotePlayer will be responsible for two tasks:
  - Reporting OUR moves to our opponent;
  - Reporting our OPPONENT's moves to us.
- Reporting moves can be done in several ways, the one we will chose is the following: each time a move is performed, the whole game state will be sent through the network to the other player. Then, the remote game state is updated and the player which just received the new game state can then play his move, and so on.
- Information is sent through the network as characters. Our first task is then to program the methods that will allow us to represent a TicTacToe game as one-dimensional array of **char**.

In the TicTacToe class, you have to:

- Create public method called toCharArray() which returns a char array of length 10. The first 9 characters represent the three lines of the game board, and the 10th character says which player is next to play. This will be used to send the game.
- Create a synchronizeWith(char[] game) method which takes a char array representation of a game and modify the current game so that it has the same state as the received one. Once the local game has been synchronized with the remote one, **the observers must be updated**. This method will be used when a new game state is received.

Part C (The Remote Player): [55 pts]

We are now able to represent a game in such a way that it can be sent over and received from the network. Now, we have to actually send and receive it. In order to do so, you will create a RemotePlayer class which extends Player and implements Runnable. The Runnable interface is discussed in Chapter 9 of the notes. A RemotePlayer has three instance variables:

- socket: of type Socket, used to establish the network connection;
- input: of type BufferedReader, used to receive opponent's moves;
- output: of type BufferedWriter, used to send our moves to the opponent.

You have to:

- Create a private void method called initializeStreams(). This method will create the BufferedReader and the BufferedWriter from the Socket. In order to do so, you will use the constructor of BufferedReader/Writer which takes an Input/OutputStream as argument. You will get these Input/OutputStreams from the socket object.
- Create two constructors for the RemotePlayer class. One will be used for the server side, while the other one will be used for the client side. To make life simpler, in this assignment, we will adopt the following convention: **the server will play with the Xs, and the client will play with the Os**.
    - For the server side:
        - The constructor only takes one parameter: the TicTacToe game.
        - The first thing you have to do is to make a call to the superclass constructor.
        - Then, create a ServerSocket which will listen on port 1200.
        - Use that ServerSocket to initialize the **socket** instance variable. You have to use the accept() method of ServerSocket.
        - Once you got your socket, the connection has been established. You now need to initialize your BufferedReader/Writer by making a call to the initializeStreams() method.
        - Then, send the local game to the client side using the appropriate write() method of your BufferedWriter. Do not forget: you first have to translate the game into an array of chars.
        - To force the immediate sending of the game, use the flush() method of the BufferedWriter.
        - The last thing you must do is to create a Thread for your server. Create a new Thread using the constructor which takes a Runnable object as argument. The Runnable you will use is your RemotePlayer (hint: you will have to use the **this** keyword). Once your Thread is created, start it.
    - For the client side:
        - The constructor takes two parameters: the TicTacToe game and the IP address of the server. This IP address should be received as a String.
        - The first thing you have to do is to make a call to the superclass constructor.
        - Then, initialize your **socket** instance variable by creating a Socket object which will connect on port 1200 of the server (identified by its IP address).
        - Once this is done, initialize your streams, make a call to the synchronizeWithRemoteGame() method (to be defined), and create a new Thread the same way you did for the server side. The call to synchronizeWithRemoteGame() is to make sure both players agree on who is playing first.

- We now have to specify how we will receive the opponent's moves. In order to do it, create a synchronizeWithRemoteGame() method. In this method, you have to:
    - Create a char array of length 10.
    - Read 10 characters from the BufferedReader. These characters have to be stored in the array you created above.
    - Then, to avoid an infinite loop, unregister the player as an observer to the game.
    - Synchronize the game with the array-represented game you just received.
    - Re-register the player as an observer.
- Good! But when are we gonna call the method we just created? In the run() method we have to define in order to be runnable. Because we started a new thread, this method will always run in background. So, in the run() method, do the following:
    - As long as the game is not finished, synchronize with the remote game.
    - When the game is finished, **close the socket**.
- Done? This was at most 10 lines of code... So I hope it did not take you too long! Now, we only have one thing left to do... send OUR moves to the opponent! This will be done in the update() method of the remote player. Each time a new move is performed, this method is called. So we will take advantage of the occasion to send the game with its new state. In order to do so, you have to:
    - First make sure that the socket is not closed, if it is, then just return and do nothing.
    - If the socket is open, then convert the game into an array of characters and send it to the opponent using the write method of the BufferedWriter. Once again, you have to force the information to be sent immediately using the flush() method of the BufferedWriter.
- The handleClick() method of the RemotePlayer does nothing.

Part D (Updating the Controller): [10 pts]

We are almost done! All we now have left to do is to allow the user to play against a remote opponent. In order to do it, all we have to do is to update the createPlayer() method of the TicTacToe panel. Instead of choosing between a human and an AI player, the user will now choose between a human, an AI and a remote player. Do not forget our convention: **the server will play with the Xs, and the client will play with the Os**. Depending on which player you are creating, you will use either the constructor for the server side or the constructor for the client side. The constructor for the client side needs an IP address: you will ask it to the user by mean of the showInputDialog() method of JOptionPane. **The default IP address has to be 127.0.0.1**. This IP address is the loopback address. This will allow you to test your application without actually using the network. It will allow you to locally connect on your server. So, to test your application, you have to:

- Launch a first instance of your program;
- Select a remote player for the Xs (this will create a server);
- Launch a second instance of your program;
- Select a human player for the Xs;
- Select a remote player for the Os (this will create a client);
- Connect to the IP address 127.0.0.1 (this will locally connect to your server);
- The first application you launched will then ask you for the type of player you want for the Os, select human player.
- And you are done! The moves you are playing on one application should actually appear on both! You are now ready for online TicTacToe gaming!

Submission

As usual, you will submit to the Raven system. You should submit an entire directory that contains all of your java files, your class files and any testing/output files that you created. Also, have a readme.txt file that clearly indicates which files are which.

Good Programming and Good Practice Requirements.

These requirements pertain regardless of what your application is supposed to do (i.e. regardless of the design requirements). These requirements are to ensure that your code is readable and maintainable by other programmers (or TA's in our case), and that your program is robust (It does not crash from bad object references). You will loose 5 marks from your total assignment mark for each of the following requirements that are not satisfied. If you do not satisfy requirement R0 you will get nothing for the assignment.

R0) IMPORTANT Uniqueness Requirement. The solution and code you submit MUST be unique. That is, it cannot be a copy, or be too similar, to someone else's assignment, or other code found elsewhere. A mark of 0 will be assigned to any assignment that is judged by us or the TA's not to be unique.

R1) All of your variables, methods and classes should have meaningful names that reflect their purpose. Do not follow the convention common in math courses where they say things like: "let x be the number of customers and let y be the number of products...". Instead call your variables numberOfCustomers or numberOfProducts. Your program should not have any variables called "x" unless there is a good reason for them to be called "x". (One exception: It's OK to call simple for-loop counters i, j and k etc. when the context is clear.)

R2) All variables in your classes should be private, unless a specific design requirements asks for them to be public (which is unlikely). We will design objects that provide services to others through their public methods. How they store their variables is their own private business.

R3) Robustness Requirements: Your program should never crash because of a "null pointer exception". This exception means that you are using a variable that does not actually refer to an object. We instruct the TA's to try and crash your program for this reason so guard against it.

R4) Comments in your code must coincide with what the code actually does. It is a very common bug in industry for people to modify code and forget to modify the comments and so you end up with comments that say one thing and code that actually does another. (By the way, try not to over-comment your code; instead choose good variable names and method names which make the code more "self commenting".)

R5) Java 1.5 Requirement: Your code should compile with the Java 1.5SDK without warnings.