CSS 543
Program 3: Online Tic-Tac-Toe Game

Professor: Munehiro Fukuda

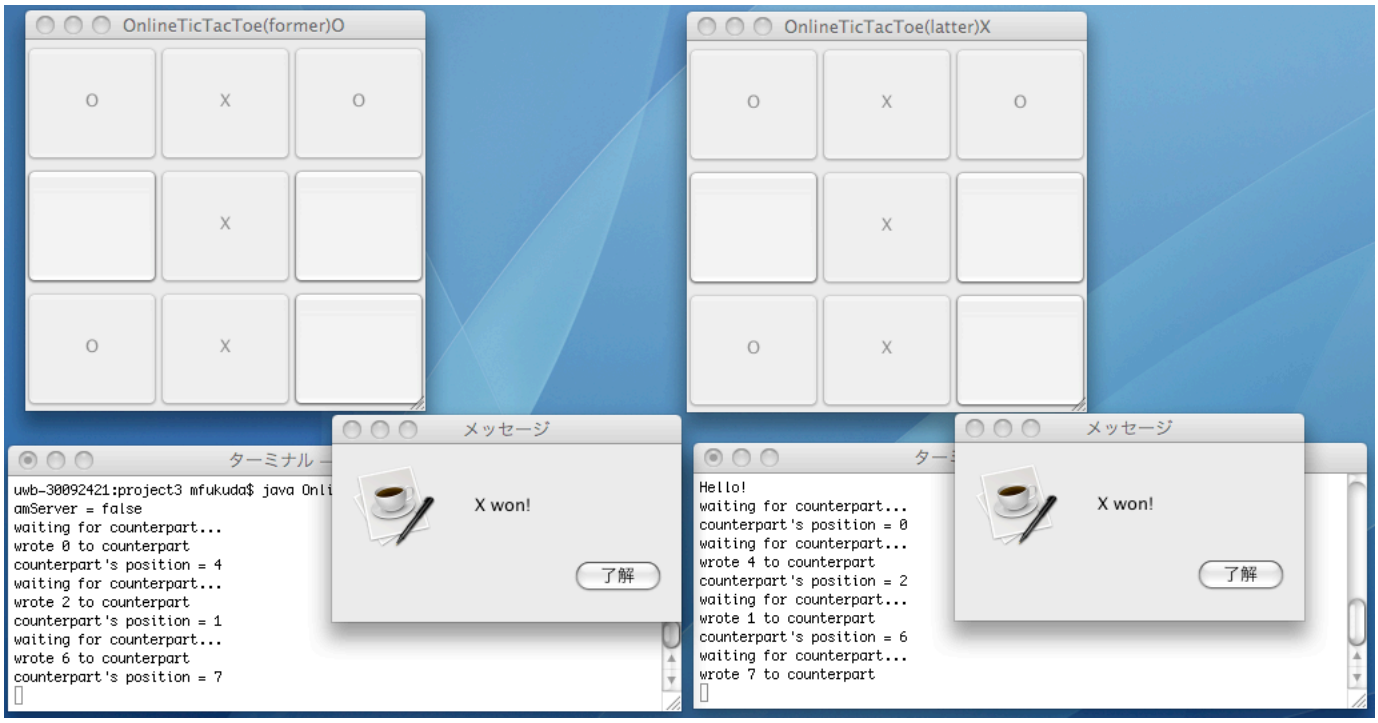
Due date: see the syllabus

1. Purpose

This assignment exercises how to write a peer-to-peer communicating program using non-blocking accept( ) as well as multiple threads, (specifically saying, the main and the slave threads).

2. Online Tic Tac Toe

This is an online Internet program that involves two users in the same tic-tac-toe game. Each user starts the game locally and operates on the local 3-by-3 tic-tac-toe window that however interacts with his/her remote counterpart's window through the Internet, so that the two users can view the same ongoing progress in their game.



The game itself is as simple as needless to say. Therefore, the following specifications focus on only how to start the program and how to manage the game window:

(a) Each of two users starts a game with his/her counterpart's IP address and port as follows:
```
java OnlineTicTacToe uw1-320-10 12345
```
They do not care which of their programs would behave as a TCP client and a server. The users may be sitting at the same computer, which thus allows the IP address to be "localhost".

(b) After (at most) one TCP connection has been established, each program must decide which will play first with the mark "O" and second with the mark "X".

(c) The graphics window marked with "O" must play first, thus accepting its user's choice of nine buttons. The selected button is marked with "O", which should be reflected to the same button on the counterpart's graphics window. Similarly, the graphics window marked with "X" must play second, mark its user's choice of nine buttons with "X", and reflect it to the counterpart's graphics window.

(d) While the counterpart is playing, the local user cannot click any button of his/her game window. Such an action must be ignored or postponed until the local user gets a turn to play. Ignoring or postponing a too early action is up to your design.
(e) Every time a user (regardless of local or remote) clicks a button, your program needs to check if the user has won the current game, in which case a winning message such as "O won!" or "X won!" should come out on the monitor.

3. Graphics

This online tic-tac-toe game needs to display a 3-by-3 graphics window with which the local user can play. Since the main purpose of this programming project is peer-to-peer communication using non-blocking accept and multithreads, you do not have to spend too much time for graphics. Below, I will show you a framework of the program that gives a complete definition of methods regarding graphics.

<table>
<tr><th>Methods</th><th>Descriptions</th></tr>
<tr><td>main(String[] args)</td><td>Verifies the correctness of two arguments, each including the counterpart's IP address and port respectively, and thereafter instantiates an OnlineTicTacToe object.

No need to modify main( ).</td></tr>
<tr><td>OnlineTicTacToe(InetAddress addr, int port)</td><td>Is the OnlineTicTacToe's constructor that sets up a TCP connection with the counterpart, brings up a game window, and starts a slave thread for listening to the counterpart.

You need to modify OnlineTicTacToe( ) to set up at most one TCP connection to the counterpart with addr and port, using non-blocking accept( ). After a connection establishment, the constructor must makeWindow(true) at one player site as well as makeWindow(false) at the other player site, in which way makeWindow(true) prompts the corresponding user to play first with mark "O" whereas makeWindow(false) indicates that the corresponding user should play second with mark "X".</td></tr>
<tr><td>makeWindow(boolean amFormer)</td><td>Pops up a graphical window with which a user can play a tic-tac-toe game. If the argument is true, the window displays "TicTacToe(Former) O", otherwise "TicTacToe(Latter) X".

You don't have to modify it. Just use it as it is.</td></tr>
<tr><td>markButton(int i, String mark)</td><td>Marks the i-th button with a given mark ("O" or "X"). The placement of nine buttons is:
<table><tr><td>0</td><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td><td>5</td></tr><tr><td>6</td><td>7</td><td>8</td></tr></table>
No need to modify it. Just use it as it is.</td></tr>
<tr><td>int whichButtonClicked(ActionEvent event)</td><td>Given an ActionEvent object (passed from actionPerformed( ) which you must modify), whichButtonClicked( ) returns the integer value corresponding to the button just clicked. See markButton( ) for the placement of nine buttons.

No need to modify it. Just use it as it is.</td></tr>
</table>

Boolean buttonMarkedWith(int i, String mark)	Returns true if the *i*-th button has been marked with a given *mark* ("O" or "X"). No need to modify it. Just use it as it is.
showWon(String mark)	Pops out a small window that indicates a winning message: "O won!" or "X won!". No need to modify it. Just use it as it is.
actionPerformed(ActionEvent event)	Represents the local user's action. It is called every time the local user clicks a button on the game window. You should call whichButtonClicked(event) so as to check which button was clicked. Use markButton( ), buttonMarkedWith( ), and showWon( ) to mark a new button with the local user's mark "O" or "X" and to check if s/he won the game. You must also inform the counterpart of which button was clicked by the local user.
Counterpart.run( )	Is the body of the Counterpart thread that represents the counterpart user's action. This thread should keep reading the established TCP connection in order to check which button was clicked by the counterpart. Similar to actionPerformed( ), this run( ) method should call markButton( ), buttonMarkedWith( ), and showWon( ) for the counterpart.

In summary, you must modify the three methods: OnlineTicTacToe( ), actionPerformed( ), and run( ) in order to complete this program.

```
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.EOFException;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketTimeoutException;
import java.net.UnknownHostException;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
/**
 *
 * @author Munehiro Fukuda
 */
public class OnlineTicTacToe implements ActionListener {

    private final int INTERVAL = 1000;          // 1 second
    private final int NBUTTONS = 9;             // #bottons
    private ObjectInputStream input = null;     // input from my counterpart
    private ObjectOutputStream output = null;   // output from my counterpart
    private JFrame window = null;               // the tic-tac-toe window
    private JButton[] button = new JButton[NBUTTONS]; // button[0] - button[9]
    private boolean[] myTurn = new boolean[1]; // T: my turn, F: your turn
    private String myMark = null;               // "O" or "X"
    private String yourMark = null;             // "X" or "O"

    /**
     * Prints out the usage.
     */
    private static void usage( ) {
        System.err.
            println( "Usage: java OnlineTicTacToe ipAddr ipPort(>=5000)" );
        System.exit( -1 );
    }
```

```
/**
 * Prints out the track trace upon a given error and quits the application.
 * @param an exception
 */
private static void error( Exception e ) {
    e.printStackTrace();
    System.exit(-1);
}

/**
 * Starts the online tic-tac-toe game.
 * @param args[0]: my counterpart's ip address, args[0]: his/her port
 */
public static void main( String[] args ) {
    // verify the number of arguments
    if ( args.length != 2 ) {
        usage( );
    }

    // verify the correctness of my counterpart address
    InetAddress addr = null;
    try {
        addr = InetAddress.getByName( args[0] );
    } catch ( UnknownHostException e ) {
        error( e );
    }

    // verify the correctness of my counterpart port
    int port = 0;
    try {
        port = Integer.parseInt( args[1] );
    } catch (NumberFormatException e) {
        error( e );
    }
    if ( port < 5000 ) {
        usage( );
    }

    // now start the application
    OnlineTicTacToe game = new OnlineTicTacToe( addr, port );
}

/**
 * Is the constructor that sets up a TCP connection with my counterpart,
 * brings up a game window, and starts a slave thread for listenning to
 * my counterpart.
 * @param my counterpart's ip address
 * @param my counterpart's port
 */
 public OnlineTicTacToe( InetAddress addr, int port ) {
    // set up a TCP connection with my counterpart

    // set up a window
    makeWindow( true ); // or makeWindow( false );

    // start my counterpart thread
    Counterpart counterpart = new Counterpart( );
    counterpart.start();
}

/**
 * Creates a 3x3 window for the tic-tac-toe game
 * @param true if this window is created by the 1st player, false by
 *        the 2nd player
 */
private void makeWindow( boolean amFormer ) {
    myTurn[0] = amFormer;
    myMark = ( amFormer ) ? "O" : "X";    // 1st person uses "O"
    yourMark = ( amFormer ) ? "X" : "O";  // 2nd person uses "X"

    // create a window
    window = new JFrame("OnlineTicTacToe(" +
            ((amFormer) ? "former)" : "latter)" ) + myMark );
    window.setSize(300, 300);
    window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
    window.setLayout(new GridLayout(3, 3));

    // initialize all nine cells.
    for (int i = 0; i < NBUTTONS; i++) {
        button[i] = new JButton();
        window.add(button[i]);
        button[i].addActionListener(this);
    }

    // make it visible
    window.setVisible(true);
}

/**
 * Marks the i-th button with mark ("O" or "X")
 * @param the i-th button
 * @param a mark ( "O" or "X" )
 * @param true if it has been marked in success
 */
 private boolean markButton( int i, String mark ) {
    if ( button[i].getText( ).equals( "" ) ) {
        button[i].setText( mark );
        button[i].setEnabled( false );
        return true;
    }
    return false;
}

/**
 * Checks which button has been clicked
 * @param an event passed from AWT
 * @return an integer (0 through to 8) that shows which button has been
 *         clicked. -1 upon an error.
 */
 private int whichButtonClicked( ActionEvent event ) {
    for ( int i = 0; i < NBUTTONS; i++ ) {
        if ( event.getSource( ) == button[i] )
            return i;
    }
    return -1;
}

/**
 * Checks if the i-th button has been marked with mark( "O" or "X" ).
 * @param the i-th button
 * @param a mark ( "O" or "X" )
 * @return true if the i-th button has been marked with mark.
 */
private boolean buttonMarkedWith( int i, String mark ) {
    return button[i].getText( ).equals( mark );
}

/**
 * Pops out another small window indicating that mark("O" or "X") won!
 * @param a mark ( "O" or "X" )
 */
private void showWon( String mark ) {
    JOptionPane.showMessageDialog( null, mark + " won!" );
}

/**
 * Is called by AWT whenever any button has been clicked. You have to:
 * <ol>
 * <li> check if it is my turn,
 * <li> check which button was clicked with whichButtonClicked( event ),
 * <li> mark the corresponding button with markButton( buttonId, mark ),
 * <li> check which button was clicked with whichButtonClicked( event ),
 * <li> mark the corresponding button with markButton( buttonId, mark ),
 * <li> send this informatioin to my counterpart,
 * <li> checks if the game was completed with
 *      buttonMarkedWith( buttonId, mark )
 * <li> shows a winning message with showWon( )
 */
public void actionPerformed( ActionEvent event ) {
    // Implement by yourself
}
```

```
    /**
     * This is a reader thread that keeps reading fomr and behaving as my
     * counterpart.
     */
    private class Counterpart extends Thread {

        /**
         * Is the body of the Counterpart thread.
         */
        @Override
        public void run( ) {
            // Implement by yourself
        }
    }
}
```

The above code is also available at **~css543/programming/project3/OnlineTicTacToeFrame.java**. You may copy it into your own working directory and modify it to complete your assignment.

4. Statement of Work

Implement the OnlineTicTacToe.java game program as described above as well as add to your program a new feature such as:

(a) Asking players for their intention to play a new game,
(b) Switching the first and second player's turn for a new game,
(c) Playing with an automated partner, (i.e., a single-user game), or
(d) Whatever interesting feature you would like to implement

For your program development, you may use any platform such as Unix, AIX, Linux, Mac OS, Open Solaris, and Windows as far as they support Java. Verify your program using any two of uw1-320 machines. The verification should be done for the following two cases:

(a) Using two different machines
Example: from uw1-320-10, type java OnlineTicTacToe uw1-320-11 12345
from uw1-320-11, type java OnlineTicTacToe uw1-320-10 12345
(b) Using a single machine
Example: from two terminal windows at uw1-320-10, type
java OnlineTicTacToe localhost 12345 or
java OnlineTicTacToe uw1-320-10 12345

5. What to Turn in

This programming assignment is due at the beginning of class on the due date. Please turn in the following materials in a hard copy. No email submission is accepted.

Criteria	Grade
Documentation of your implementation of the OnlineTicTacToe program in one or two pages. Insufficient or too much (i.e., less than 1 page or more than 2 pages) documentation receives 4pts.	5pts
Source code that adheres good modularization, coding style, and an appropriate amount of commends. 5pts: well-organized and correct code using synchronized/wait/notify keywords 4pts: messy yet working code or code with spin loops (i.e., without using synchronized/wait/notify keywords) 3pts: code with wrong synchronization (regardless of using spin loops or synchronized/wait/notify keywords) 2pts: incomplete code	5pts

Execution output that verifies the correctness of all your implementation. 5pts: correct snapshots of your program execution 4pts: snapshots with minor errors or insufficient snapshots 3pts: wrong synchronization 2pts: incomplete execution	5pts
Discussions about additional features, limitations, and possible improvements of your program in one or two pages (firm). 5pts: Good discussions with clearly mentioning the new feature(s) you added to the TicTacToe game 4pts: Good discussions but no new feature added to the original TicTacToe specification. 3pts: Unremarkable discussions without any new feature added to the original TicTacToe specification.	5pts
Lab Sessions 5 through 6 counts 1pt for each. If you have not yet turned in a hard copy of your source code/output or missed any session(s), please turn in together with program 3.	2pts
Total Note that program 3 takes 15% of your final grade.	22pts