CSS 533
Program 1: Online Tic-Tac-Toe Game

Aim: This assignment is about developing Tic-tac-toe game using a peer-to-peer communicating program using non-blocking accept(), multiple threads, (specifically saying, the main and the slave threads), and JSCH (Java secure shell).

Documentation:

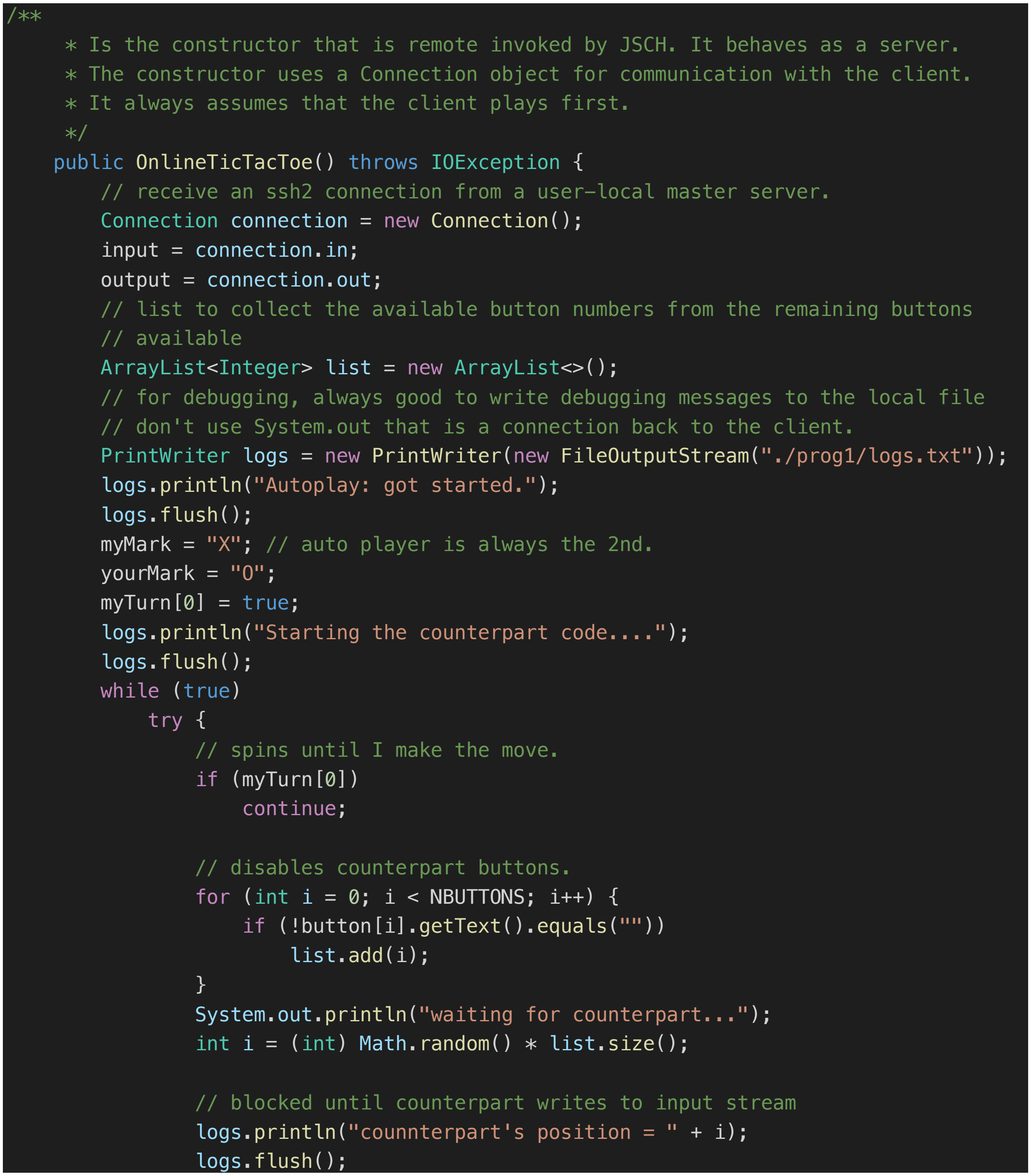
- The peer-to-peer communication is using non-blocking accept(), multiple threads is written in OnlineTicTacToe (InetAddress addr, int port) for interactive play between two users on the same machine as well as on the different machines.
- The counterpart executes the second player where the myTurn variable is synchronized between the client and the server to play only when they have turns.
- In two user interactive play the host machine checks if it his turn and marks the button on the window with myMark and passes the information to the counterpart using output stream object. It also checks whether the game is finished using checkGame() method.
- The checkGame() method takes in the mark passed to it and checks if the same marks are present on the diagonally, vertically and horizontally. If yes it sends true and else false. This method also checks if there is a tie between two players by traversing through all the buttons and if the buttons are not empty and the traversing count is equal to the number of buttons then there is a tie.
- If the game is over the playAgain window is executed from the winning counterpart. If the user says yes, then the game window is cleared and all the buttons on the both windows are enabled. If user says no then it displays the message “thanks for playing”.
- To connect as a localhost on the same machine using same port, the peer connection is done after the server tries to accept the connection and it fails then it throws a Exception where the connection to the peer is done using Inetaddress and port and again the accept() is called where it accepts the connection and says that it is a server.
- The Input and Output stream for both client and server are created in the constructor OnlineTicTacToe (InetAddress addr, int port).
- The auto option enables the user to have only one window and play like “pass-n-play” option in which only one user at a time can play on the window.
- The auto option is enabled using jsch library where the connection to the remote machine is done via Connection class which uses the exec command to execute the following command on the remote machine with hostname

```
String command = "java -cp " + cur_dir + "/jsch-0.1.54.jar:" + cur_dir + " OnlineTicTacToe";
```

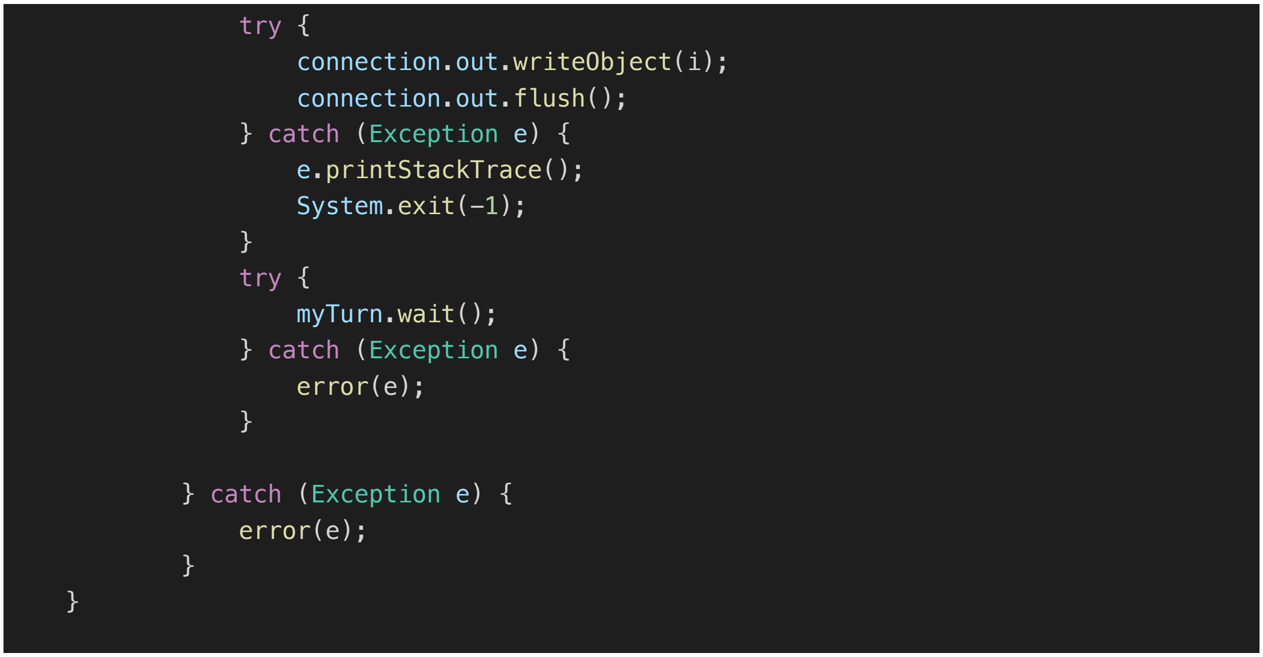
- The communication with the remote machine is possible using logs.txt file for writing the commands from the remote machine to the host machine and remote machine marks random button from the available choices.

Source code:

(1) OnlineTicTacToe( ):

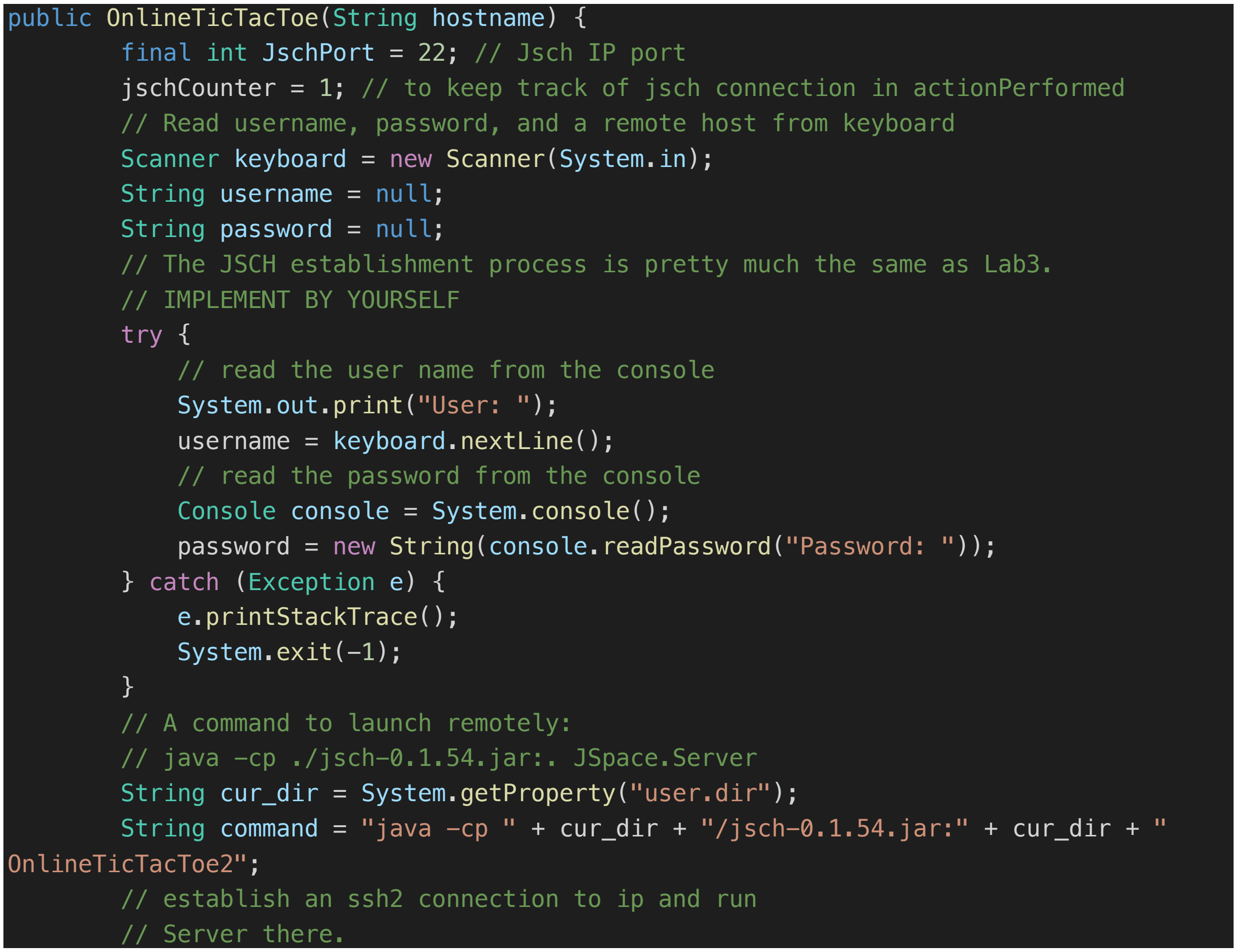


```
/**
     * Is the constructor that is remote invoked by JSCH. It behaves as a server.
     * The constructor uses a Connection object for communication with the client.
     * It always assumes that the client plays first.
     */
    public OnlineTicTacToe() throws IOException {
        // receive an ssh2 connection from a user-local master server.
        Connection connection = new Connection();
        input = connection.in;
        output = connection.out;
        // list to collect the available button numbers from the remaining buttons
        // available
        ArrayList<Integer> list = new ArrayList<>();
        // for debugging, always good to write debugging messages to the local file
        // don't use System.out that is a connection back to the client.
        PrintWriter logs = new PrintWriter(new FileOutputStream("./prog1/logs.txt"));
        logs.println("Autoplay: got started.");
        logs.flush();
        myMark = "X"; // auto player is always the 2nd.
        yourMark = "O";
        myTurn[0] = true;
        logs.println("Starting the counterpart code....");
        logs.flush();
        while (true)
            try {
                // spins until I make the move.
                if (myTurn[0])
                    continue;

                // disables counterpart buttons.
                for (int i = 0; i < NBUTTONS; i++) {
                    if (!button[i].getText().equals(""))
                        list.add(i);
                }
                System.out.println("waiting for counterpart...");
                int i = (int) Math.random() * list.size();

                // blocked until counterpart writes to input stream
                logs.println("counnterpart's position = " + i);
                logs.flush();
```

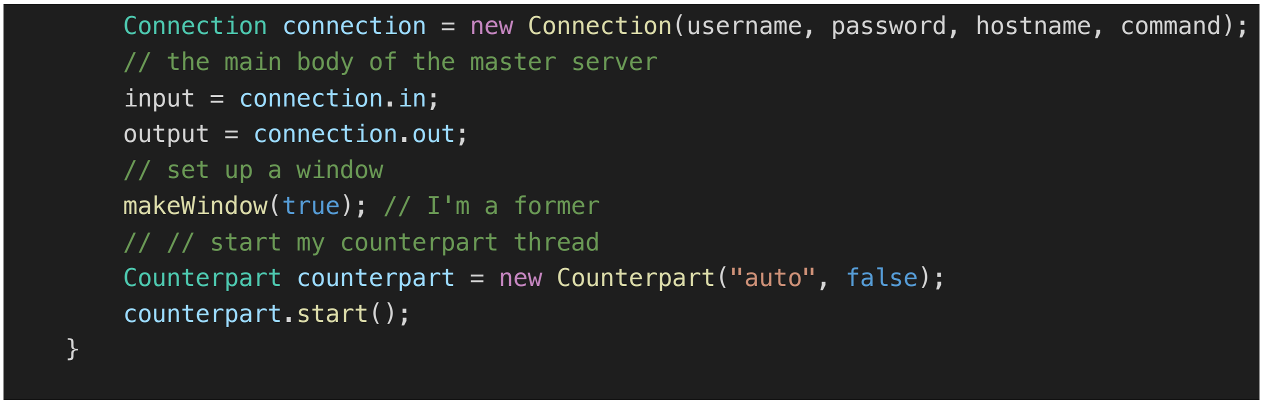


```
                try {
                    connection.out.writeObject(i);
                    connection.out.flush();
                } catch (Exception e) {
                    e.printStackTrace();
                    System.exit(-1);
                }
                try {
                    myTurn.wait();
                } catch (Exception e) {
                    error(e);
                }

            } catch (Exception e) {
                error(e);
            }
    }
```

(2) OnlineTicTacToe(String hostname):



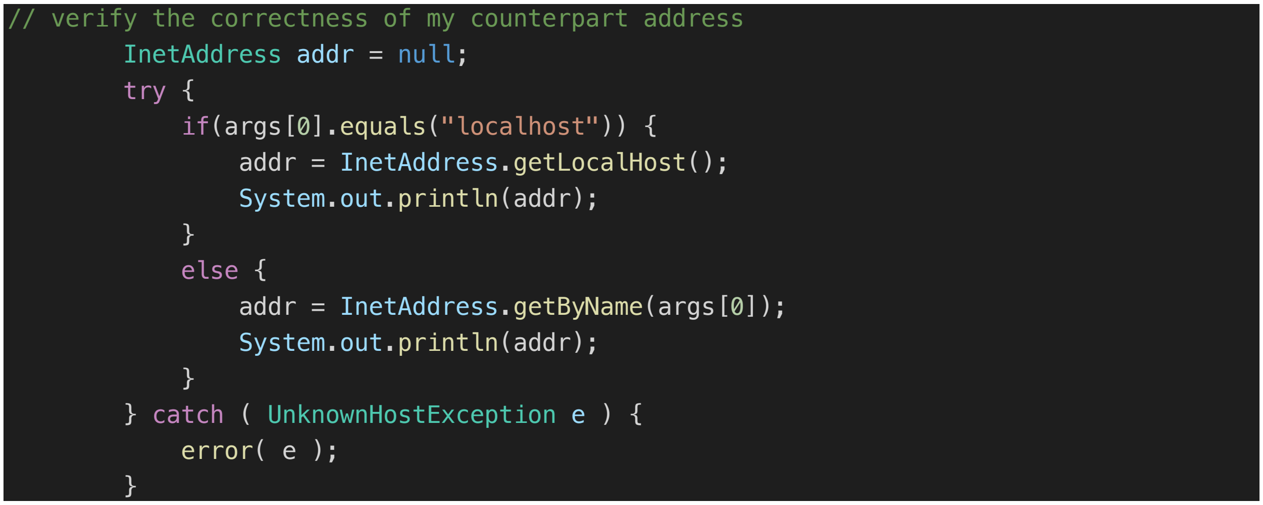
```
public OnlineTicTacToe(String hostname) {
        final int JschPort = 22; // Jsch IP port
        jschCounter = 1; // to keep track of jsch connection in actionPerformed
        // Read username, password, and a remote host from keyboard
        Scanner keyboard = new Scanner(System.in);
        String username = null;
        String password = null;
        // The JSCH establishment process is pretty much the same as Lab3.
        // IMPLEMENT BY YOURSELF
        try {
            // read the user name from the console
            System.out.print("User: ");
            username = keyboard.nextLine();
            // read the password from the console
            Console console = System.console();
            password = new String(console.readPassword("Password: "));
        } catch (Exception e) {
            e.printStackTrace();
            System.exit(-1);
        }
        // A command to launch remotely:
        // java -cp ./jsch-0.1.54.jar:. JSpace.Server
        String cur_dir = System.getProperty("user.dir");
        String command = "java -cp " + cur_dir + "/jsch-0.1.54.jar:" + cur_dir + "
OnlineTicTacToe2";
        // establish an ssh2 connection to ip and run
        // Server there.
```



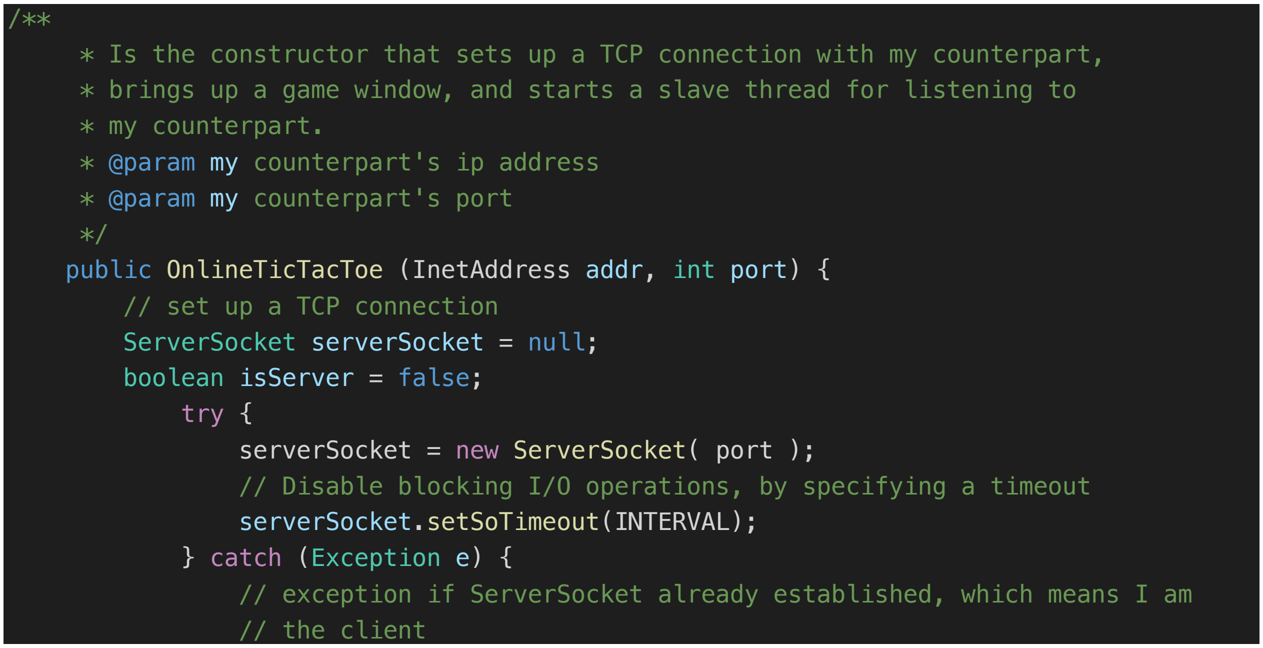
```
        Connection connection = new Connection(username, password, hostname, command);
        // the main body of the master server
        input = connection.in;
        output = connection.out;
        // set up a window
        makeWindow(true); // I'm a former
        // // start my counterpart thread
        Counterpart counterpart = new Counterpart("auto", false);
        counterpart.start();
    }
```

(3) OnlineTicTacToe(InetAddress addr, int port):

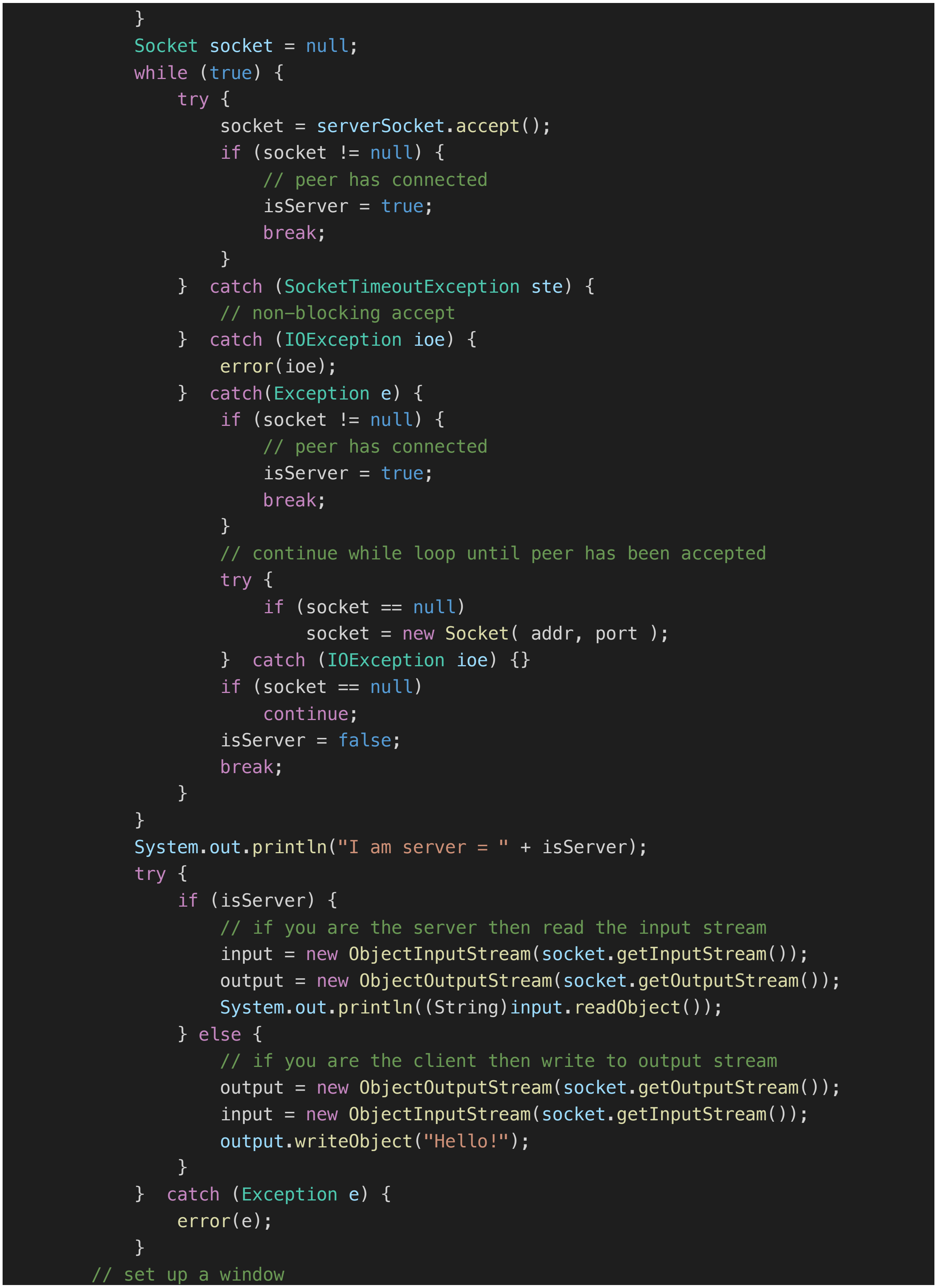
Code for connecting as a localhost:



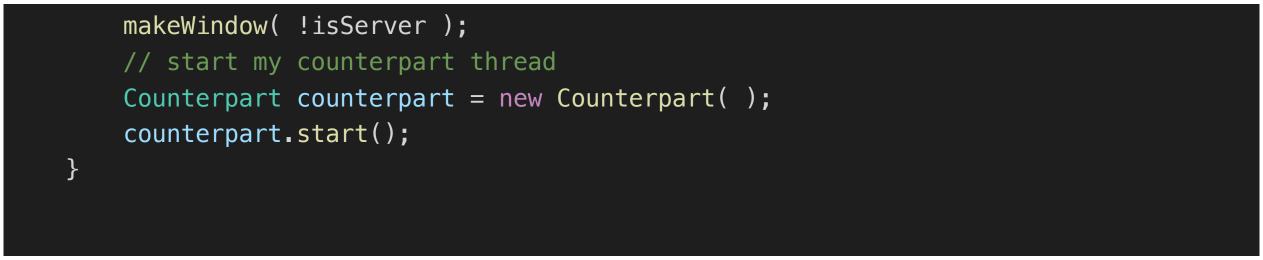
```
// verify the correctness of my counterpart address
        InetAddress addr = null;
        try {
            if(args[0].equals("localhost")) {
                addr = InetAddress.getLocalHost();
                System.out.println(addr);
            }
            else {
                addr = InetAddress.getByName(args[0]);
                System.out.println(addr);
            }
        } catch ( UnknownHostException e ) {
            error( e );
        }
```



```
/**
     * Is the constructor that sets up a TCP connection with my counterpart,
     * brings up a game window, and starts a slave thread for listening 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
        ServerSocket serverSocket = null;
        boolean isServer = false;
            try {
                serverSocket = new ServerSocket( port );
                // Disable blocking I/O operations, by specifying a timeout
                serverSocket.setSoTimeout(INTERVAL);
            } catch (Exception e) {
                // exception if ServerSocket already established, which means I am
                // the client
```

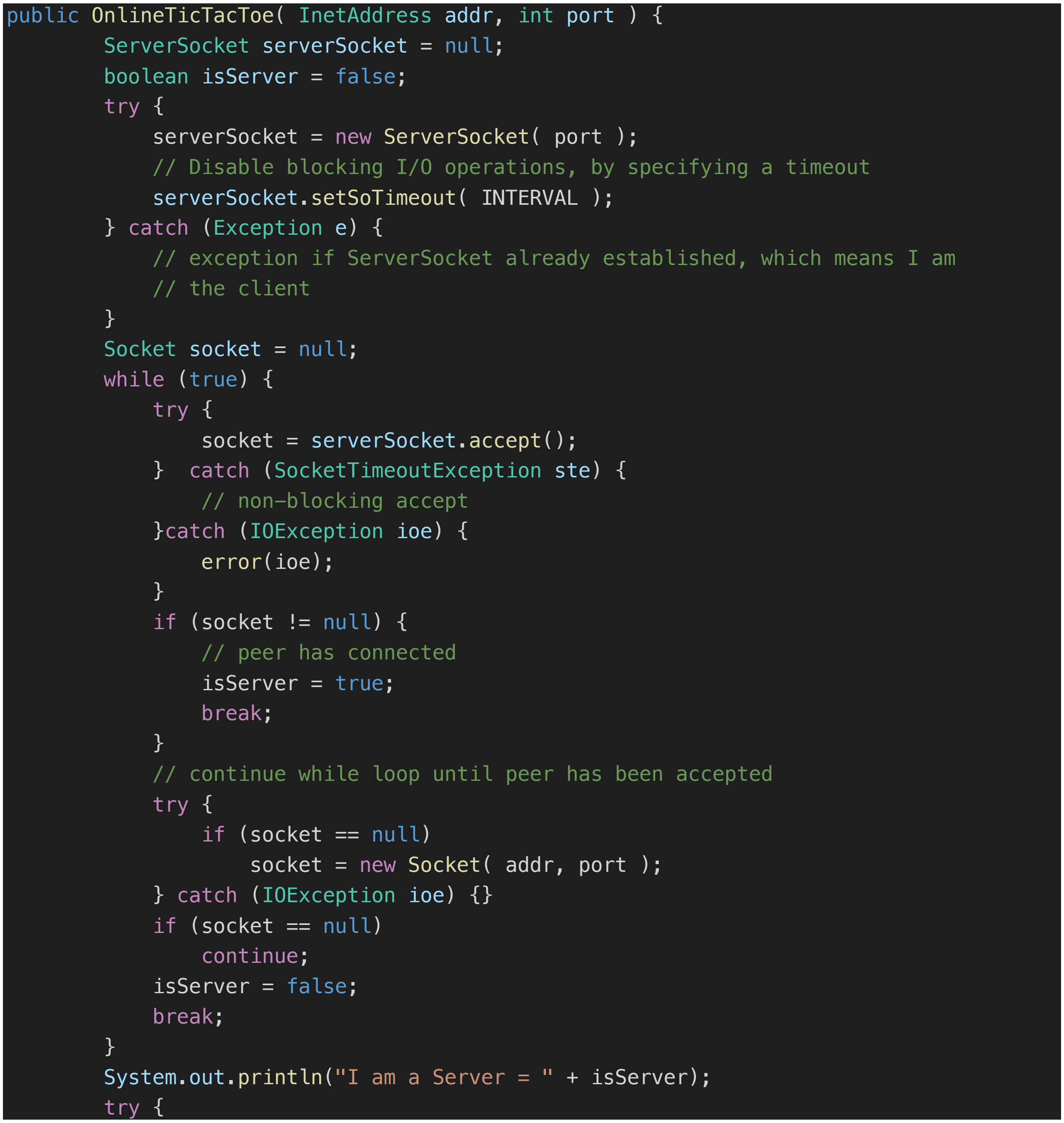


```java
            }
            Socket socket = null;
            while (true) {
                try {
                    socket = serverSocket.accept();
                    if (socket != null) {
                        // peer has connected
                        isServer = true;
                        break;
                    }
                }  catch (SocketTimeoutException ste) {
                    // non-blocking accept
                }  catch (IOException ioe) {
                    error(ioe);
                }  catch(Exception e) {
                    if (socket != null) {
                        // peer has connected
                        isServer = true;
                        break;
                    }
                    // continue while loop until peer has been accepted
                    try {
                        if (socket == null)
                            socket = new Socket( addr, port );
                    }  catch (IOException ioe) {}
                    if (socket == null)
                        continue;
                    isServer = false;
                    break;
                }
            }
            System.out.println("I am server = " + isServer);
            try {
                if (isServer) {
                    // if you are the server then read the input stream
                    input = new ObjectInputStream(socket.getInputStream());
                    output = new ObjectOutputStream(socket.getOutputStream());
                    System.out.println((String)input.readObject());
                } else {
                    // if you are the client then write to output stream
                    output = new ObjectOutputStream(socket.getOutputStream());
                    input = new ObjectInputStream(socket.getInputStream());
                    output.writeObject("Hello!");
                }
            }  catch (Exception e) {
                error(e);
            }
        // set up a window
```

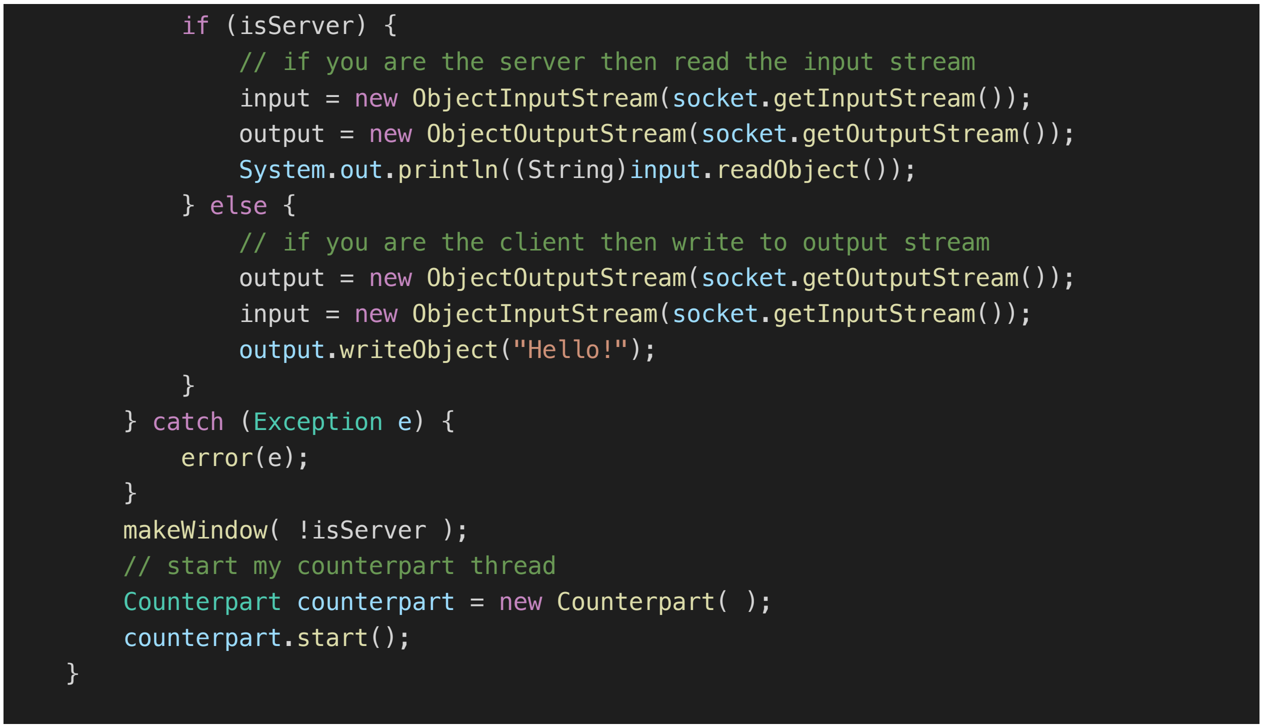


```
        makeWindow( !isServer );
        // start my counterpart thread
        Counterpart counterpart = new Counterpart( );
        counterpart.start();
    }
```

(3) OnlineTicTacToe(InetAddress addr, int port):
Code for connecting on two different machines:

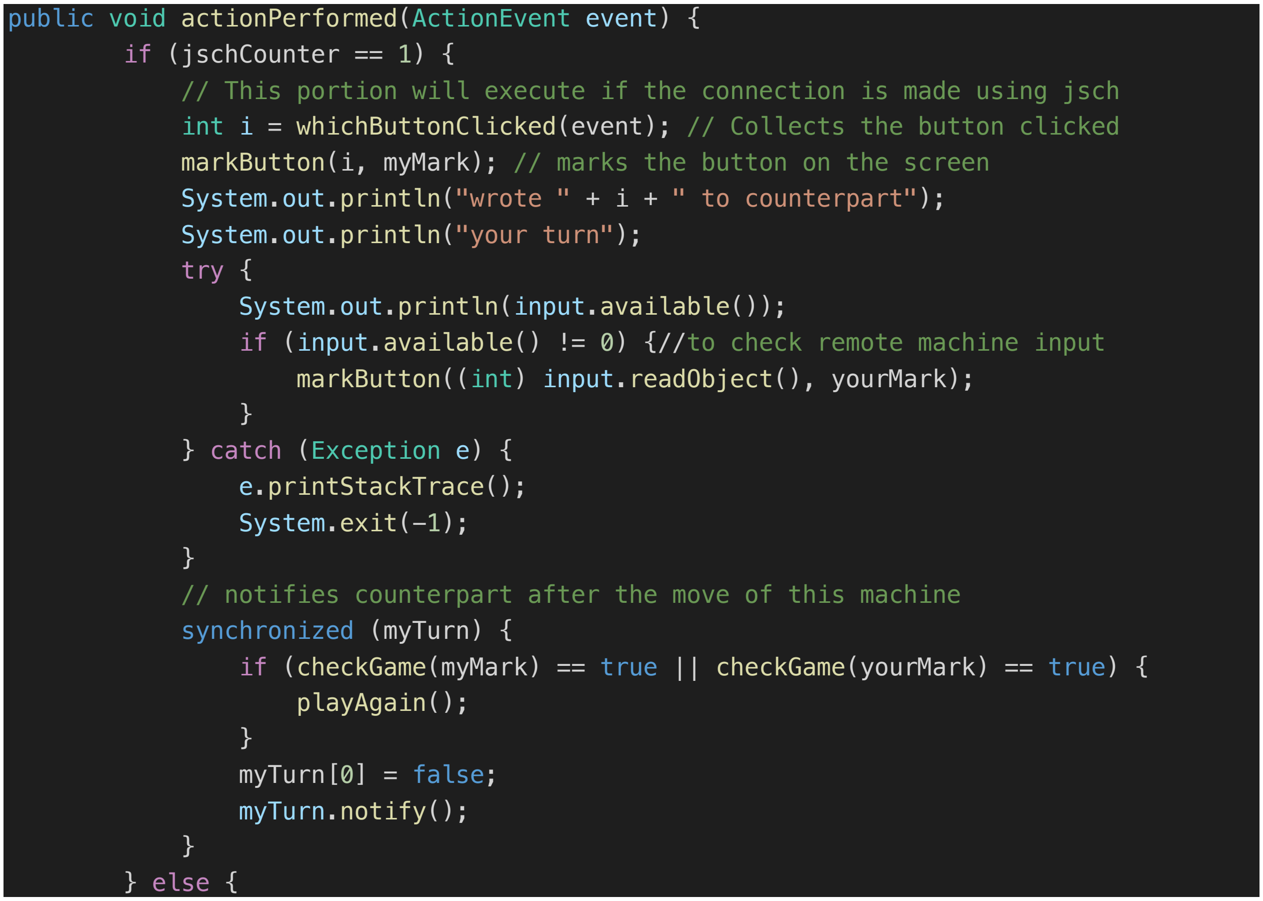


```
public OnlineTicTacToe( InetAddress addr, int port ) {
        ServerSocket serverSocket = null;
        boolean isServer = false;
        try {
            serverSocket = new ServerSocket( port );
            // Disable blocking I/O operations, by specifying a timeout
            serverSocket.setSoTimeout( INTERVAL );
        } catch (Exception e) {
            // exception if ServerSocket already established, which means I am
            // the client
        }
        Socket socket = null;
        while (true) {
            try {
                socket = serverSocket.accept();
            }  catch (SocketTimeoutException ste) {
                // non-blocking accept
            }catch (IOException ioe) {
                error(ioe);
            }
            if (socket != null) {
                // peer has connected
                isServer = true;
                break;
            }
            // continue while loop until peer has been accepted
            try {
                if (socket == null)
                    socket = new Socket( addr, port );
            } catch (IOException ioe) {}
            if (socket == null)
                continue;
            isServer = false;
            break;
        }
        System.out.println("I am a Server = " + isServer);
        try {
```

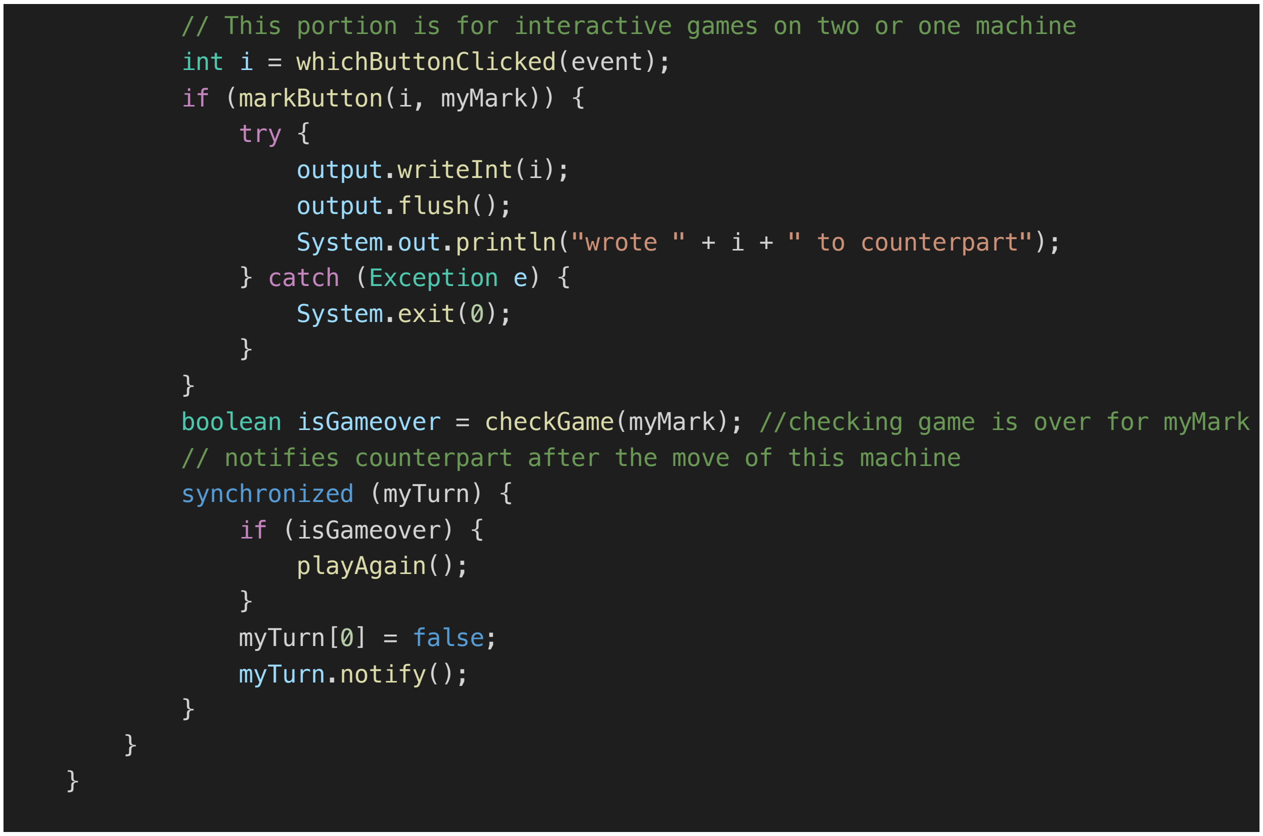


```java
            if (isServer) {
                // if you are the server then read the input stream
                input = new ObjectInputStream(socket.getInputStream());
                output = new ObjectOutputStream(socket.getOutputStream());
                System.out.println((String)input.readObject());
            } else {
                // if you are the client then write to output stream
                output = new ObjectOutputStream(socket.getOutputStream());
                input = new ObjectInputStream(socket.getInputStream());
                output.writeObject("Hello!");
            }
        } catch (Exception e) {
            error(e);
        }
        makeWindow( !isServer );
        // start my counterpart thread
        Counterpart counterpart = new Counterpart( );
        counterpart.start();
    }
```

(4) actionPerformed(ActionEvent event):

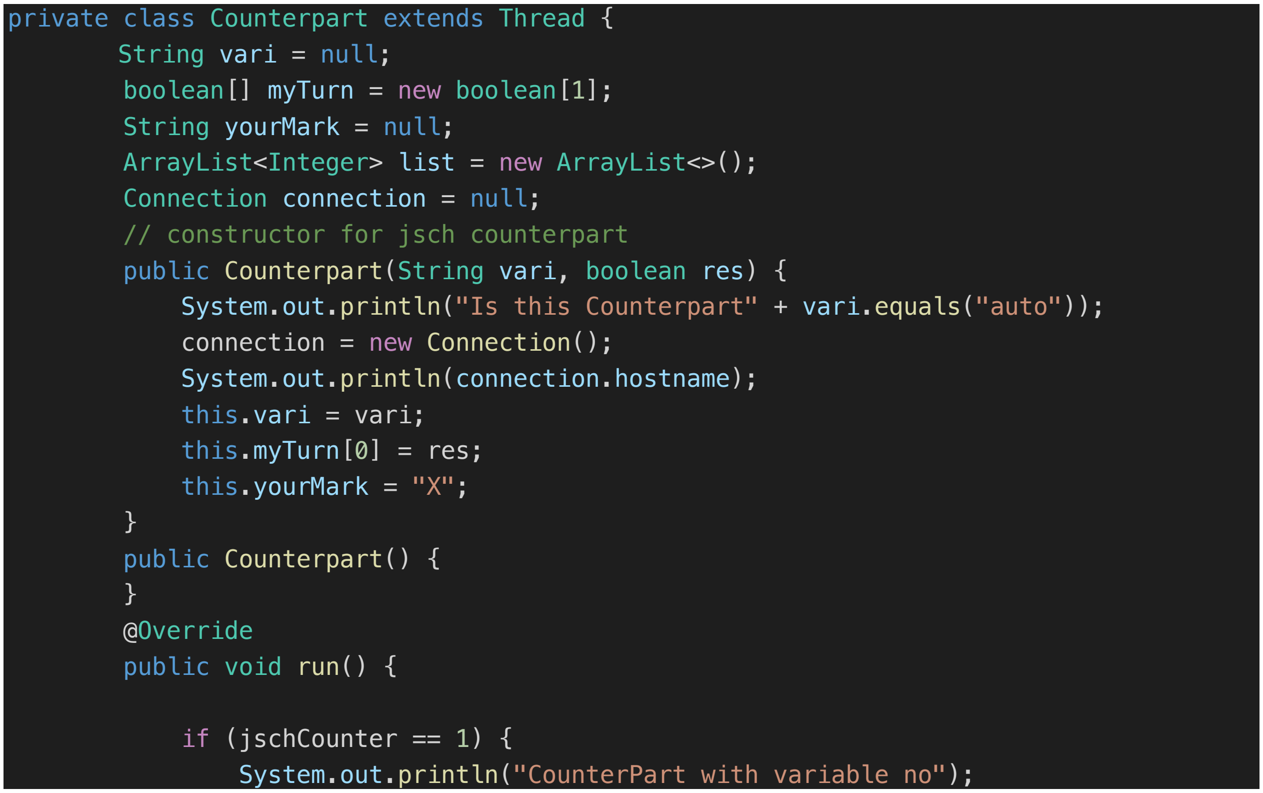


```java
public void actionPerformed(ActionEvent event) {
        if (jschCounter == 1) {
            // This portion will execute if the connection is made using jsch
            int i = whichButtonClicked(event); // Collects the button clicked
            markButton(i, myMark); // marks the button on the screen
            System.out.println("wrote " + i + " to counterpart");
            System.out.println("your turn");
            try {
                System.out.println(input.available());
                if (input.available() != 0) {//to check remote machine input
                    markButton((int) input.readObject(), yourMark);
                }
            } catch (Exception e) {
                e.printStackTrace();
                System.exit(-1);
            }
            // notifies counterpart after the move of this machine
            synchronized (myTurn) {
                if (checkGame(myMark) == true || checkGame(yourMark) == true) {
                    playAgain();
                }
                myTurn[0] = false;
                myTurn.notify();
            }
        } else {
```

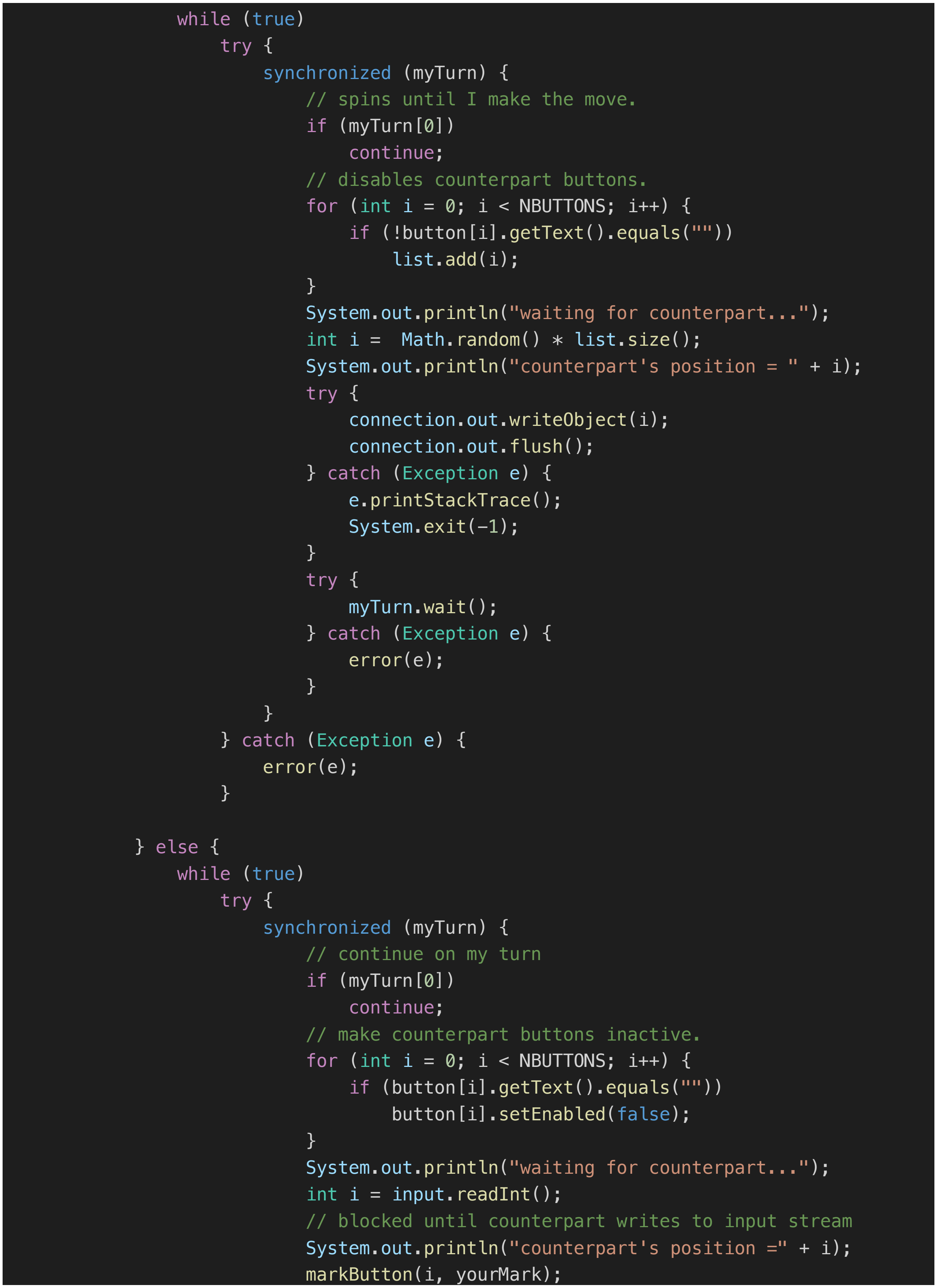


```
            // This portion is for interactive games on two or one machine
            int i = whichButtonClicked(event);
            if (markButton(i, myMark)) {
                try {
                    output.writeInt(i);
                    output.flush();
                    System.out.println("wrote " + i + " to counterpart");
                } catch (Exception e) {
                    System.exit(0);
                }
            }
            boolean isGameover = checkGame(myMark); //checking game is over for myMark
            // notifies counterpart after the move of this machine
            synchronized (myTurn) {
                if (isGameover) {
                    playAgain();
                }
                myTurn[0] = false;
                myTurn.notify();
            }
        }
    }
```

(5) Counterpart.run( ):



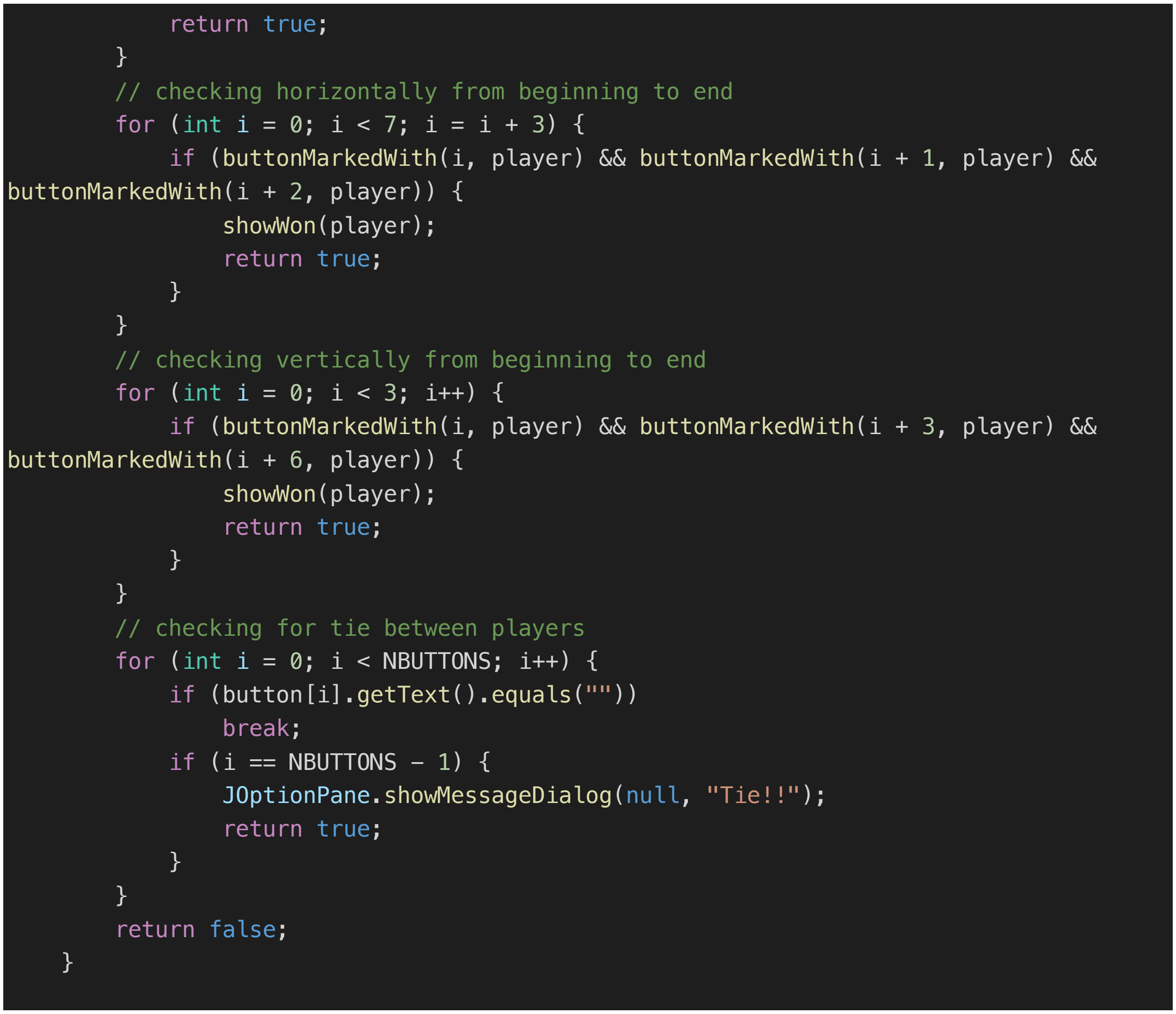
```
private class Counterpart extends Thread {
        String vari = null;
        boolean[] myTurn = new boolean[1];
        String yourMark = null;
        ArrayList<Integer> list = new ArrayList<>();
        Connection connection = null;
        // constructor for jsch counterpart
        public Counterpart(String vari, boolean res) {
            System.out.println("Is this Counterpart" + vari.equals("auto"));
            connection = new Connection();
            System.out.println(connection.hostname);
            this.vari = vari;
            this.myTurn[0] = res;
            this.yourMark = "X";
        }
        public Counterpart() {
        }
        @Override
        public void run() {

            if (jschCounter == 1) {
                System.out.println("CounterPart with variable no");
```



```
                while (true)
                    try {
                        synchronized (myTurn) {
                            // spins until I make the move.
                            if (myTurn[0])
                                continue;
                            // disables counterpart buttons.
                            for (int i = 0; i < NBUTTONS; i++) {
                                if (!button[i].getText().equals(""))
                                    list.add(i);
                            }
                            System.out.println("waiting for counterpart...");
                            int i =  Math.random() * list.size();
                            System.out.println("counterpart's position = " + i);
                            try {
                                connection.out.writeObject(i);
                                connection.out.flush();
                            } catch (Exception e) {
                                e.printStackTrace();
                                System.exit(-1);
                            }
                            try {
                                myTurn.wait();
                            } catch (Exception e) {
                                error(e);
                            }
                        }
                    } catch (Exception e) {
                        error(e);
                    }

            } else {
                while (true)
                    try {
                        synchronized (myTurn) {
                            // continue on my turn
                            if (myTurn[0])
                                continue;
                            // make counterpart buttons inactive.
                            for (int i = 0; i < NBUTTONS; i++) {
                                if (button[i].getText().equals(""))
                                    button[i].setEnabled(false);
                            }
                            System.out.println("waiting for counterpart...");
                            int i = input.readInt();
                            // blocked until counterpart writes to input stream
                            System.out.println("counterpart's position =" + i);
                            markButton(i, yourMark);
                            boolean isGameOver = checkGame(yourMark);
                            try {
                                if (!isGameOver) {
                                    // activates the other players buttons.
                                    for (int j = 0; j < NBUTTONS; j++) {
                                        if (button[j].getText().equals(""))
                                            button[j].setEnabled(true);
                                    }
                                }
                                if (isGameOver) {
                                    playAgain();
                                }
                                myTurn.wait();
                            } catch (Exception e) {
                                error(e);
                            }
                        }
                    } catch (EOFException eof) {
                        JOptionPane.showMessageDialog(null, "counterpart denied from
playing the game!");
                        System.out.println("counterpart disconnected.");
                        System.exit(0);
                    } catch (Exception e) {
                        error(e);
                    }
            }
        }
    }
```

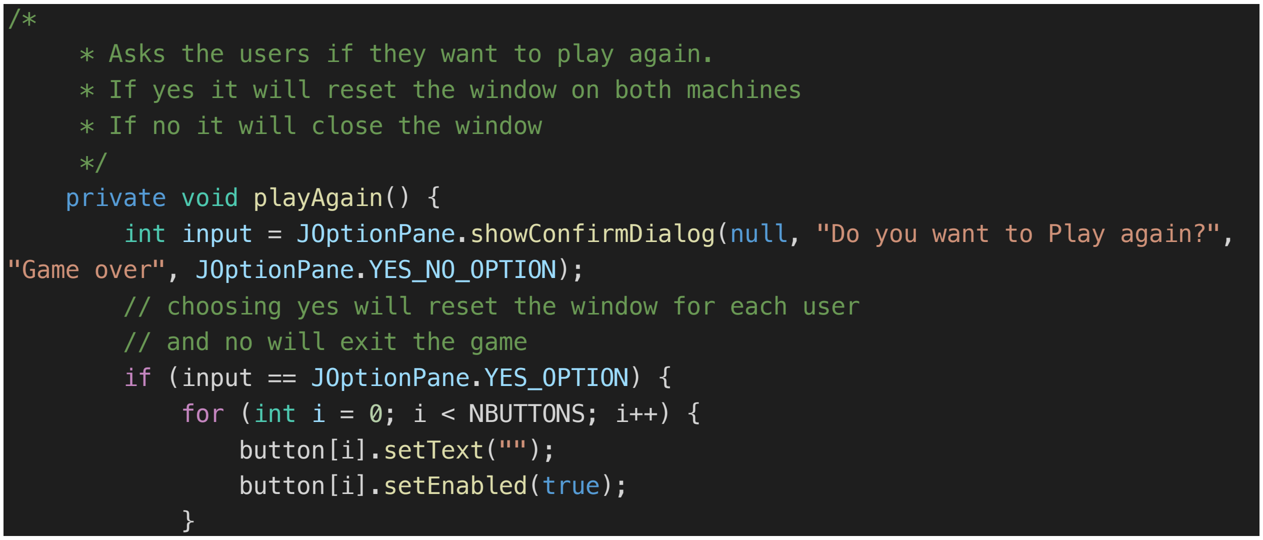
(6) CheckGame( ):

```
/**
     * checkGame will check for the winnig or tie case for each mark
     * For win it will check if the mark is adjacent diagnally,
     * horizontally and vertically
     * It also chceks for tie by counting the marked buttons
     */
    private boolean checkGame(String player) {
        // checking diagonally from left to right
        if (buttonMarkedWith(0, player) && buttonMarkedWith(4, player) &&
buttonMarkedWith(8, player)) {
            showWon(player);
            return true;
        }
        // checking diagonally from right to left
        if (buttonMarkedWith(2, player) && buttonMarkedWith(4, player) &&
buttonMarkedWith(6, player)) {
            showWon(player);
```

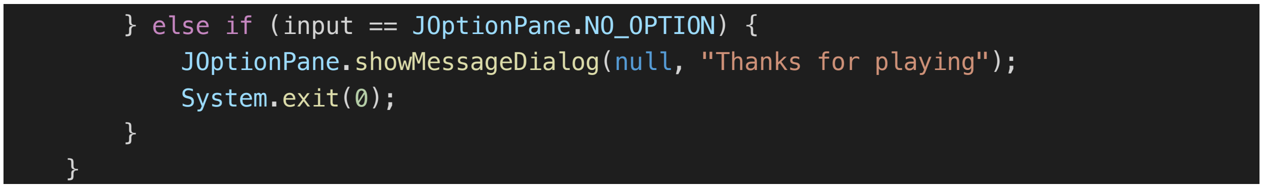


```
            return true;
        }
        // checking horizontally from beginning to end
        for (int i = 0; i < 7; i = i + 3) {
            if (buttonMarkedWith(i, player) && buttonMarkedWith(i + 1, player) &&
buttonMarkedWith(i + 2, player)) {
                showWon(player);
                return true;
            }
        }
        // checking vertically from beginning to end
        for (int i = 0; i < 3; i++) {
            if (buttonMarkedWith(i, player) && buttonMarkedWith(i + 3, player) &&
buttonMarkedWith(i + 6, player)) {
                showWon(player);
                return true;
            }
        }
        // checking for tie between players
        for (int i = 0; i < NBUTTONS; i++) {
            if (button[i].getText().equals(""))
                break;
            if (i == NBUTTONS - 1) {
                JOptionPane.showMessageDialog(null, "Tie!!");
                return true;
            }
        }
        return false;
    }
```

(6) PlayAgain( ):



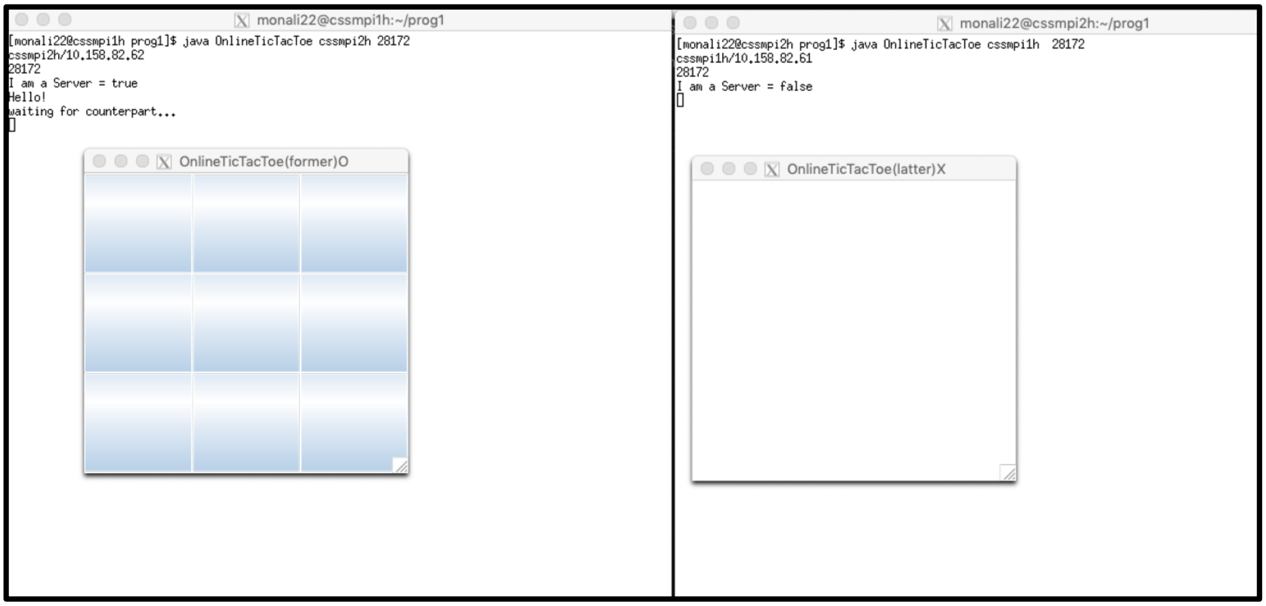
```
/*
     * Asks the users if they want to play again.
     * If yes it will reset the window on both machines
     * If no it will close the window
     */
    private void playAgain() {
        int input = JOptionPane.showConfirmDialog(null, "Do you want to Play again?",
"Game over", JOptionPane.YES_NO_OPTION);
        // choosing yes will reset the window for each user
        // and no will exit the game
        if (input == JOptionPane.YES_OPTION) {
            for (int i = 0; i < NBUTTONS; i++) {
                button[i].setText("");
                button[i].setEnabled(true);
            }
```



```
        } else if (input == JOptionPane.NO_OPTION) {
            JOptionPane.showMessageDialog(null, "Thanks for playing");
            System.exit(0);
        }
    }
```

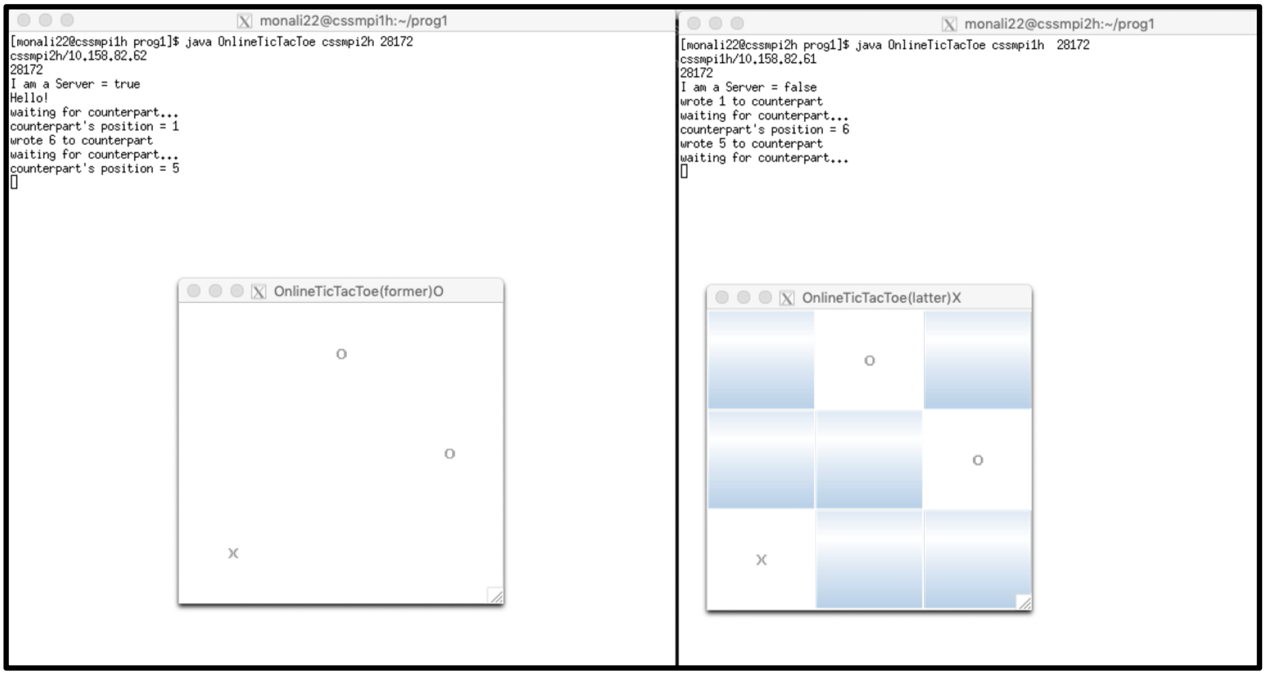
Execution output:

(1) A two-user interactive game over two computing nodes:



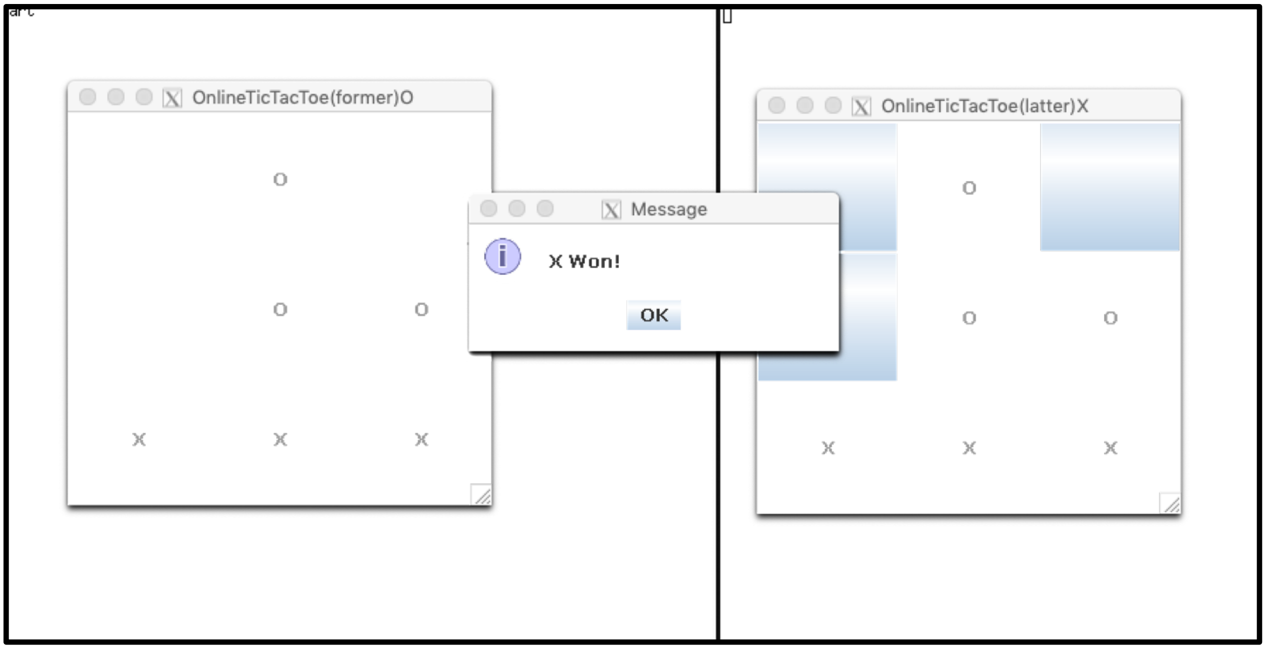
```
[monali22@cssmpi1h prog1]$ java OnlineTicTacToe cssmpi2h 28172
cssmpi2h/10.158.82.62
28172
I am a Server = true
Hello!
waiting for counterpart...
```

```
[monali22@cssmpi2h prog1]$ java OnlineTicTacToe cssmpi1h  28172
cssmpi1h/10.158.82.61
28172
I am a Server = false
```

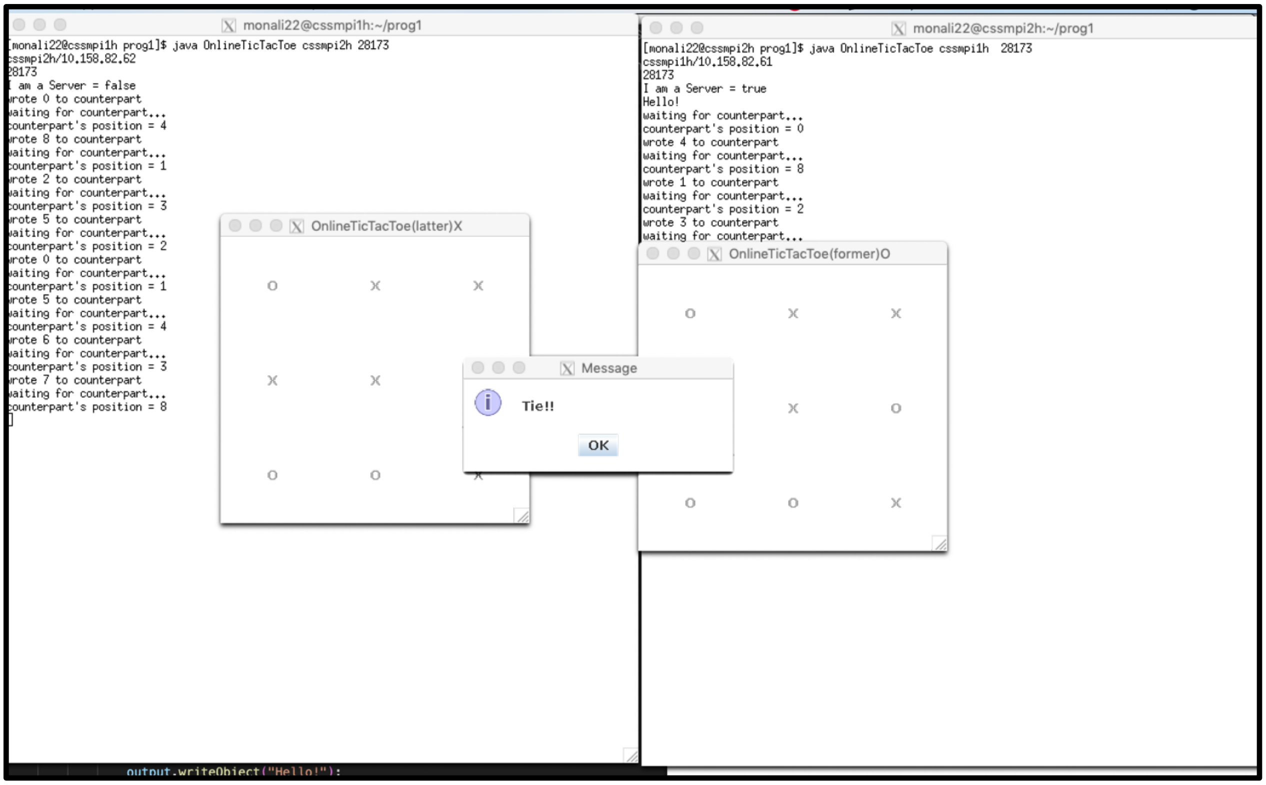


```
[monali22@cssmpi1h prog1]$ java OnlineTicTacToe cssmpi2h 28172
cssmpi2h/10.158.82.62
28172
I am a Server = true
Hello!
waiting for counterpart...
counterpart's position = 1
wrote 6 to counterpart
waiting for counterpart...
counterpart's position = 5
```

```
[monali22@cssmpi2h prog1]$ java OnlineTicTacToe cssmpi1h  28172
cssmpi1h/10.158.82.61
28172
I am a Server = false
wrote 1 to counterpart
waiting for counterpart...
counterpart's position = 6
wrote 5 to counterpart
waiting for counterpart...
```



X Won!

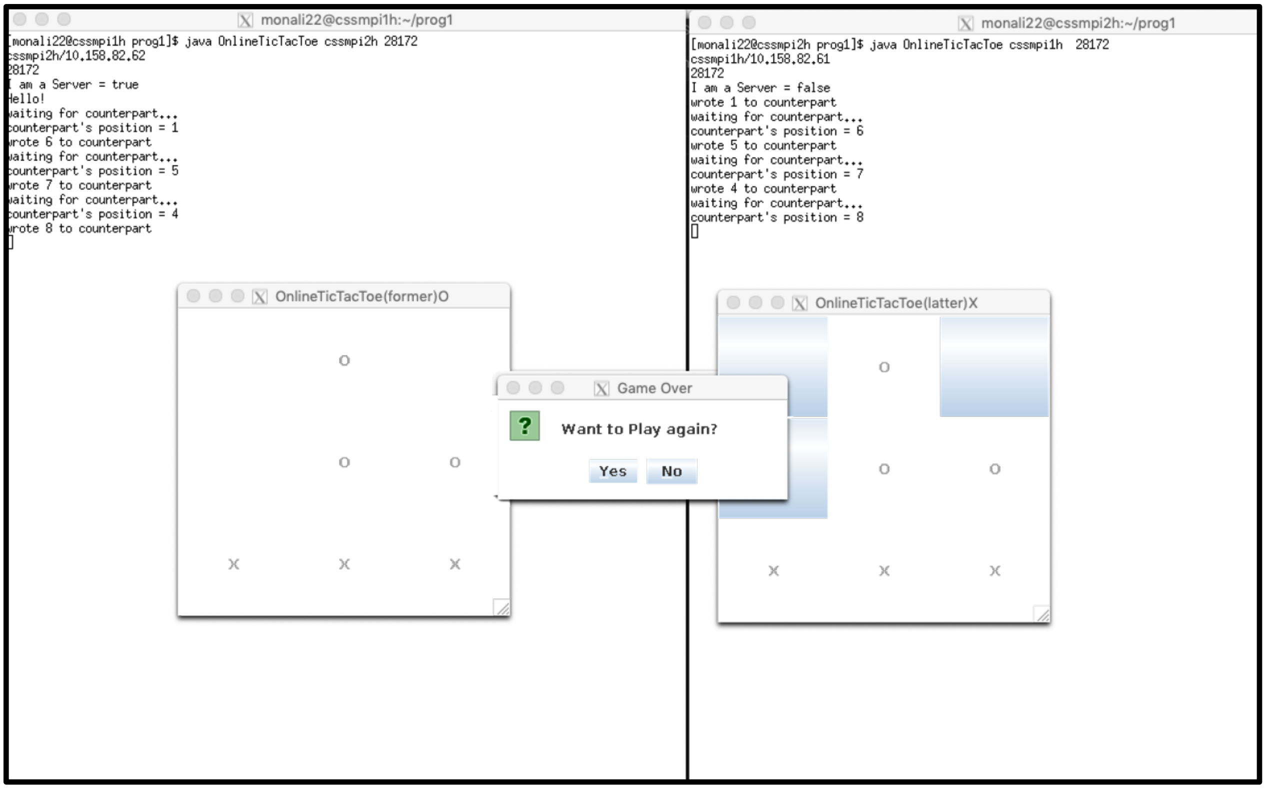


```
[monali22@cssmpi1h prog1]$ java OnlineTicTacToe cssmpi2h 28173
cssmpi2h/10.158.82.62
28173
I am a Server = false
wrote 0 to counterpart
waiting for counterpart...
counterpart's position = 4
wrote 8 to counterpart
waiting for counterpart...
counterpart's position = 1
wrote 2 to counterpart
waiting for counterpart...
counterpart's position = 3
wrote 5 to counterpart
waiting for counterpart...
counterpart's position = 2
wrote 0 to counterpart
waiting for counterpart...
counterpart's position = 1
wrote 5 to counterpart
waiting for counterpart...
counterpart's position = 4
wrote 6 to counterpart
waiting for counterpart...
counterpart's position = 3
wrote 7 to counterpart
waiting for counterpart...
counterpart's position = 8
```

```
[monali22@cssmpi2h prog1]$ java OnlineTicTacToe cssmpi1h  28173
cssmpi1h/10.158.82.61
28173
I am a Server = true
Hello!
waiting for counterpart...
counterpart's position = 0
wrote 4 to counterpart
waiting for counterpart...
counterpart's position = 8
wrote 1 to counterpart
waiting for counterpart...
counterpart's position = 2
wrote 3 to counterpart
waiting for counterpart...
```

Tie!!

```
output.writeObject("Hello!");
```

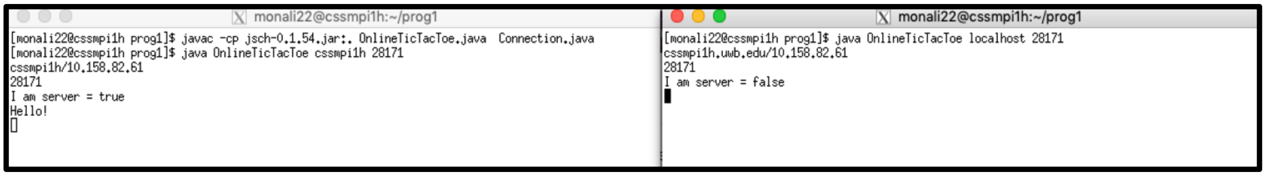


```
[monali22@cssmpi1h prog1]$ java OnlineTicTacToe cssmpi2h 28172
cssmpi2h/10.158.82.62
28172
I am a Server = true
Hello!
waiting for counterpart...
counterpart's position = 1
wrote 6 to counterpart
waiting for counterpart...
counterpart's position = 5
wrote 7 to counterpart
waiting for counterpart...
counterpart's position = 4
wrote 8 to counterpart
```

```
[monali22@cssmpi2h prog1]$ java OnlineTicTacToe cssmpi1h  28172
cssmpi1h/10.158.82.61
28172
I am a Server = false
wrote 1 to counterpart
waiting for counterpart...
counterpart's position = 6
wrote 5 to counterpart
waiting for counterpart...
counterpart's position = 7
wrote 4 to counterpart
waiting for counterpart...
counterpart's position = 8
```

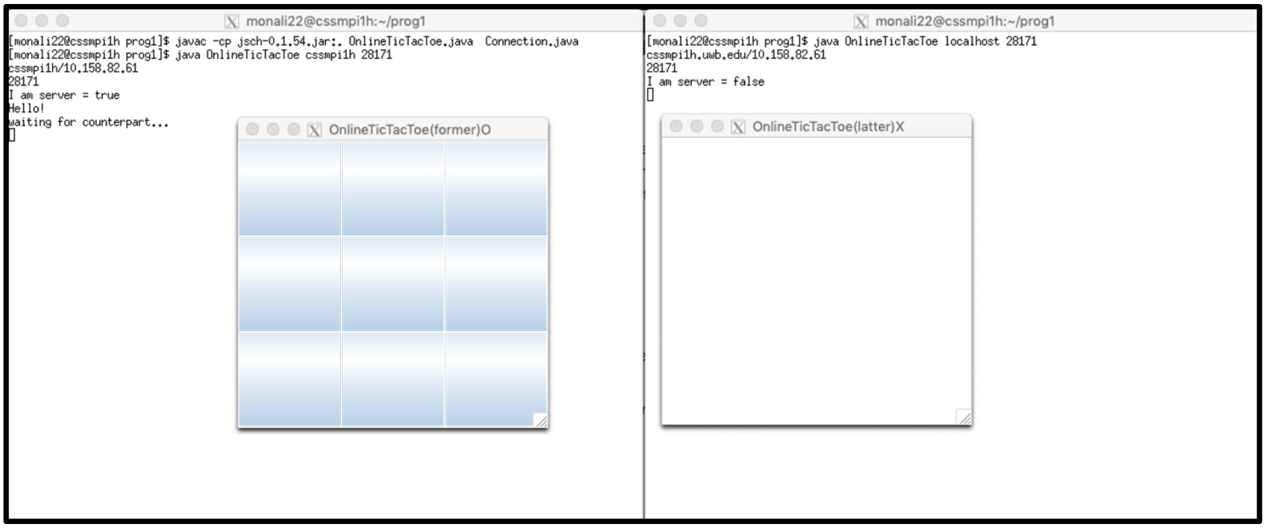
Want to Play again?

(2) (2) A two-user interactive game on a single computing node:



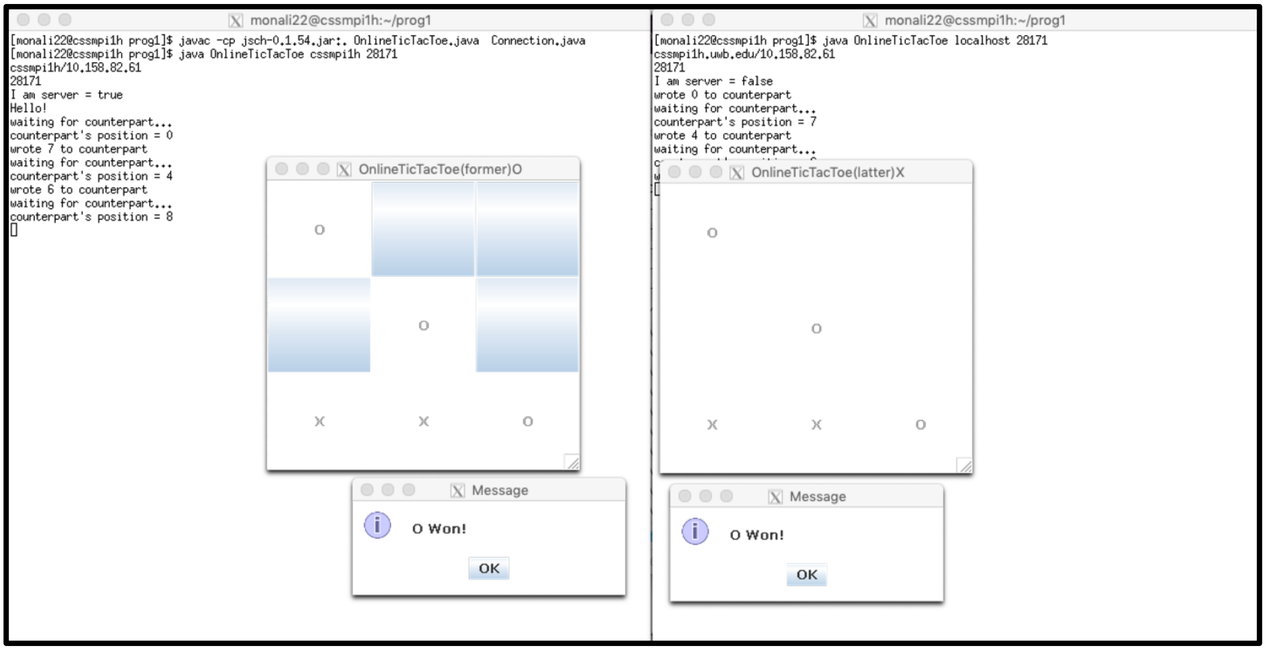
```
[monali22@cssmpi1h prog1]$ javac -cp jsch-0.1.54.jar:. OnlineTicTacToe.java  Connection.java
[monali22@cssmpi1h prog1]$ java OnlineTicTacToe cssmpi1h 28171
cssmpi1h/10.158.82.61
28171
I am server = true
Hello!
```

```
[monali22@cssmpi1h prog1]$ java OnlineTicTacToe localhost 28171
cssmpi1h.uwb.edu/10.158.82.61
28171
I am server = false
```



```
[monali22@cssmpi1h prog1]$ javac -cp jsch-0.1.54.jar:. OnlineTicTacToe.java  Connection.java
[monali22@cssmpi1h prog1]$ java OnlineTicTacToe cssmpi1h 28171
cssmpi1h/10.158.82.61
28171
I am server = true
Hello!
waiting for counterpart...
```

```
[monali22@cssmpi1h prog1]$ java OnlineTicTacToe localhost 28171
cssmpi1h.uwb.edu/10.158.82.61
28171
I am server = false
```



monali22@cssmpi1h:~/prog1

```
[monali22@cssmpi1h prog1]$ javac -cp jsch-0.1.54.jar:. OnlineTicTacToe.java  Connection.java
[monali22@cssmpi1h prog1]$ java OnlineTicTacToe cssmpi1h 28171
cssmpi1h/10.158.82.61
28171
I am server = true
Hello!
waiting for counterpart...
counterpart's position = 0
wrote 7 to counterpart
waiting for counterpart...
counterpart's position = 4
wrote 6 to counterpart
waiting for counterpart...
counterpart's position = 8
```

OnlineTicTacToe(former)O

Message

O Won!

monali22@cssmpi1h:~/prog1

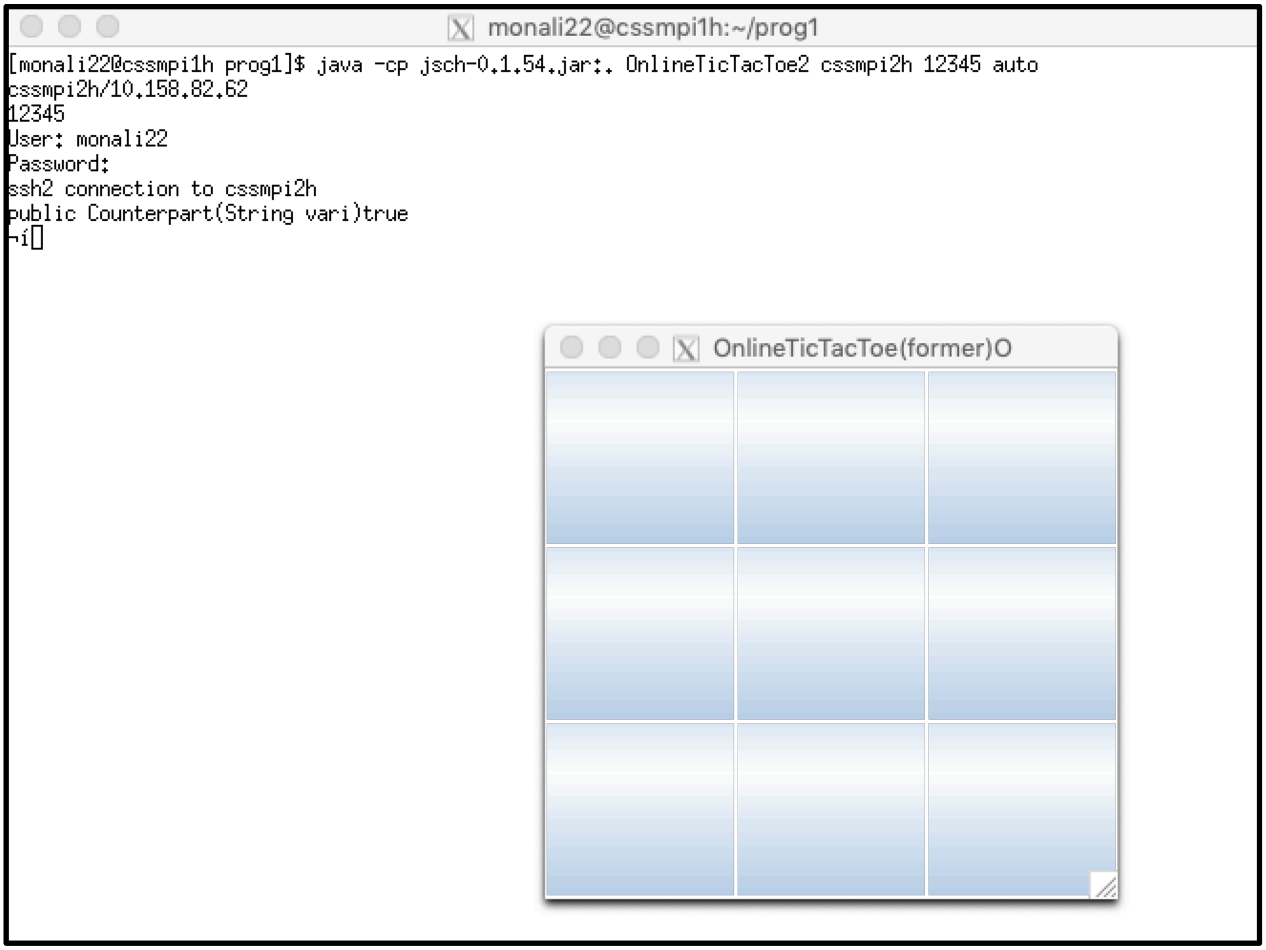
```
[monali22@cssmpi1h prog1]$ java OnlineTicTacToe localhost 28171
cssmpi1h.uwb.edu/10.158.82.61
28171
I am server = false
wrote 0 to counterpart
waiting for counterpart...
counterpart's position = 7
wrote 4 to counterpart
waiting for counterpart...
c
w
```

OnlineTicTacToe(latter)X

Message

O Won!

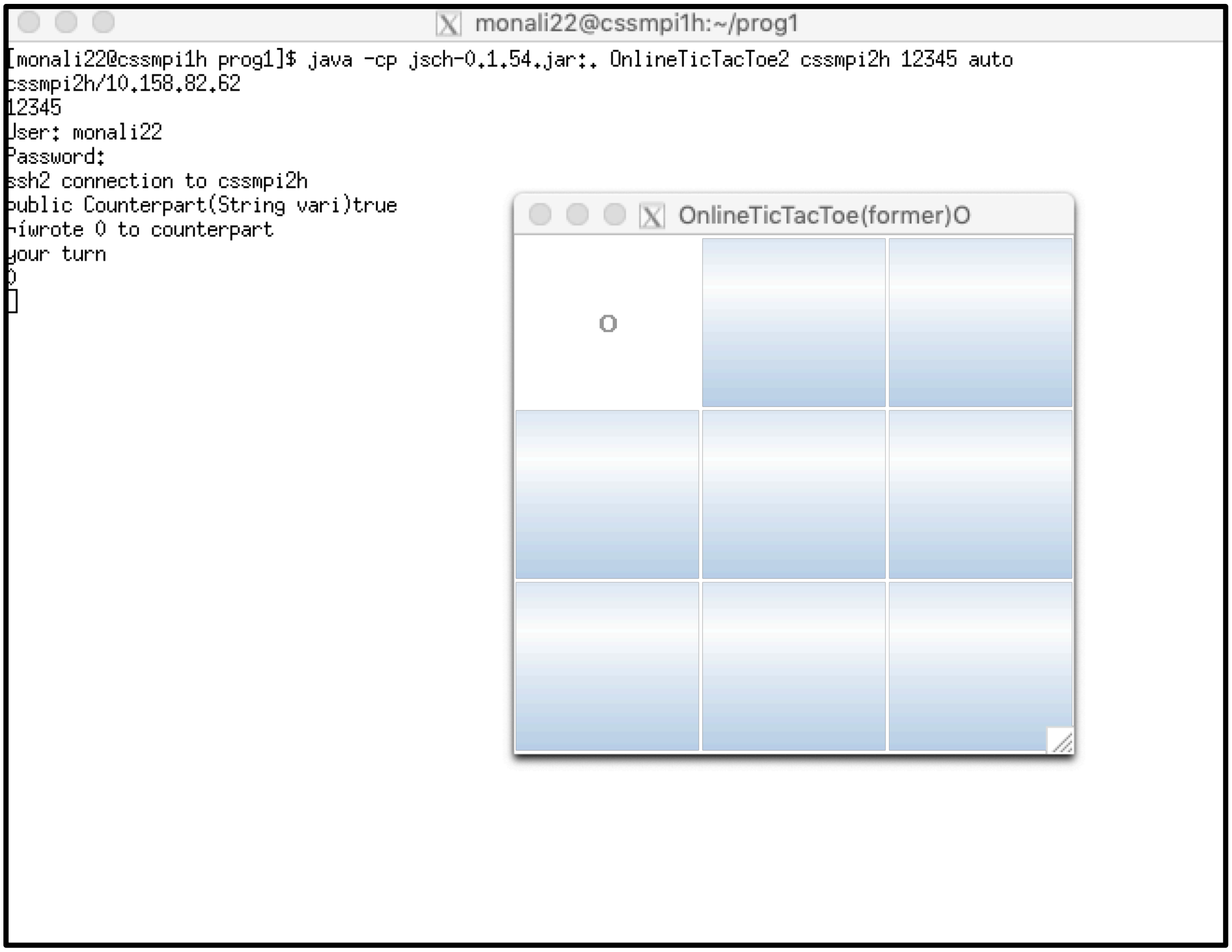
(3) A single-user automated game over two computing nodes:



monali22@cssmpi1h:~/prog1

```
[monali22@cssmpi1h prog1]$ java -cp jsch-0.1.54.jar:. OnlineTicTacToe2 cssmpi2h 12345 auto
cssmpi2h/10.158.82.62
12345
User: monali22
Password:
ssh2 connection to cssmpi2h
public Counterpart(String vari)true
¬í
```

OnlineTicTacToe(former)O



```
[monali22@cssmpi1h prog1]$ java -cp jsch-0.1.54.jar:. OnlineTicTacToe2 cssmpi2h 12345 auto
cssmpi2h/10.158.82.62
12345
User: monali22
Password:
ssh2 connection to cssmpi2h
public Counterpart(String vari)true
hiwrote 0 to counterpart
your turn
0
```

Discussions:

Additional features:

- Asking players for their intention to play a new game
- Checking for ties
- Showing denial if the one user closes the game
- Synchronization between the client and server
- Disabling the opponents screen buttons

Limitations:

- Latency in transfer of the messages between the client and server. It takes time to reach a message to the destination. Hence the destination has to wait and can't move forward with the game.
- If any issue is encountered in the synchronized block, then the other threads need to wait, and one cannot see which thread is waiting in the synchronized block.

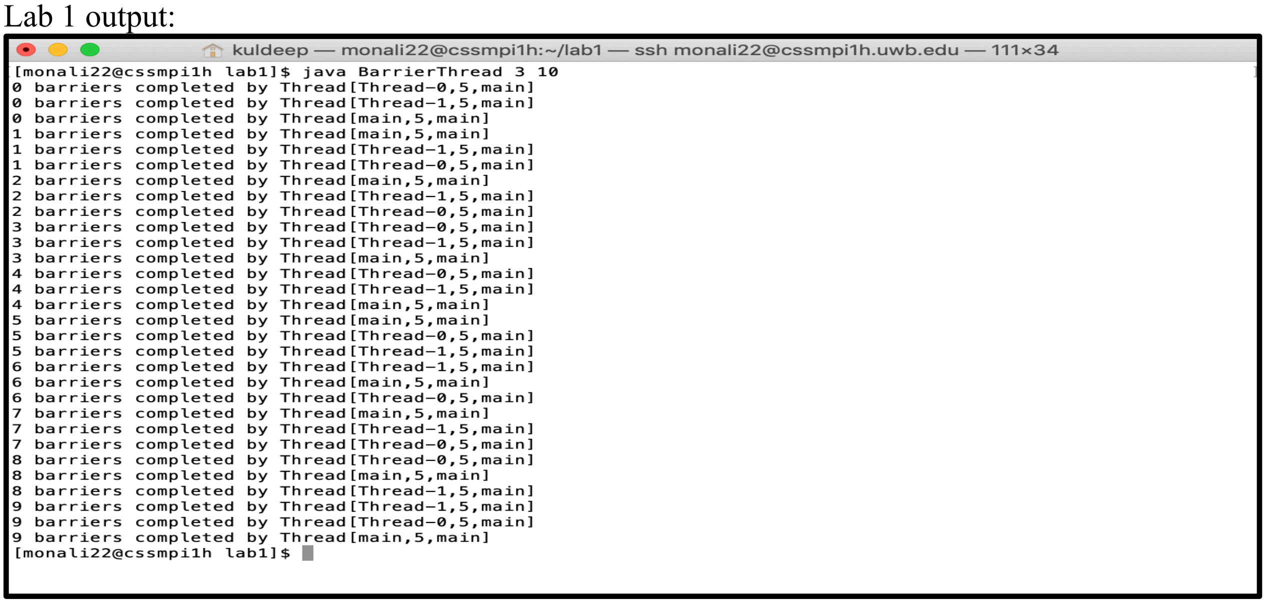
- Java synchronization does not allow concurrent reads.
- Java synchronized method run very slowly and can degrade the performance, so you should synchronize the method when it is absolutely necessary otherwise not and to synchronize block only for critical section of the code.
- When more methods synchronized under one lock and other methods under a different lock will lead to more concurrency and also increases overall performance

Possible improvements

- Locks framework in java can be implemented across the methods, allows to invoke lock() in method1 and invoke unlock() in method2.
- Trying to implement the program using **java.nio** which provides the APIs that offer features for intensive I/O operations.
- Implementation of switching the first and second player's turn for a new game by asking them
- Implementation of automated remote counterpart to make it more intelligent

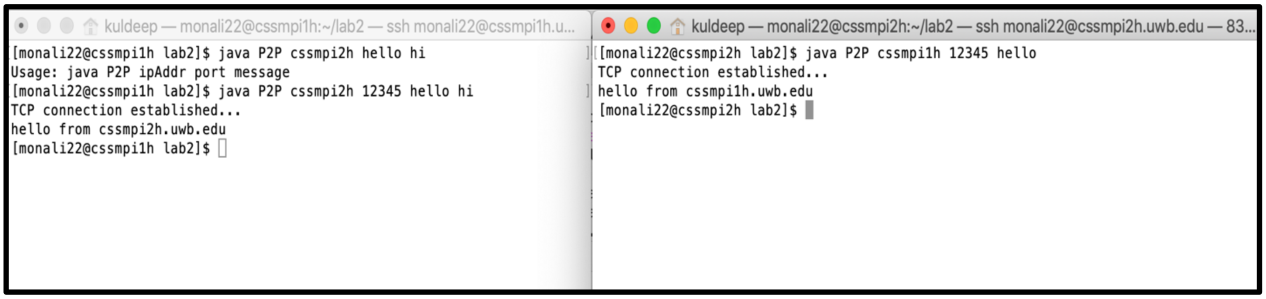
Lab Sessions 1, 2, and 3

Lab 1 output:


kuldeep — monali22@cssmpi1h:~/lab1 — ssh monali22@cssmpi1h.uwb.edu — 111×34

```
[monali22@cssmpi1h lab1]$ java BarrierThread 3 10
0 barriers completed by Thread[Thread-0,5,main]
0 barriers completed by Thread[Thread-1,5,main]
0 barriers completed by Thread[main,5,main]
1 barriers completed by Thread[main,5,main]
1 barriers completed by Thread[Thread-1,5,main]
1 barriers completed by Thread[Thread-0,5,main]
2 barriers completed by Thread[main,5,main]
2 barriers completed by Thread[Thread-1,5,main]
2 barriers completed by Thread[Thread-0,5,main]
3 barriers completed by Thread[Thread-0,5,main]
3 barriers completed by Thread[Thread-1,5,main]
3 barriers completed by Thread[main,5,main]
4 barriers completed by Thread[Thread-0,5,main]
4 barriers completed by Thread[Thread-1,5,main]
4 barriers completed by Thread[main,5,main]
5 barriers completed by Thread[main,5,main]
5 barriers completed by Thread[Thread-0,5,main]
5 barriers completed by Thread[Thread-1,5,main]
6 barriers completed by Thread[Thread-1,5,main]
6 barriers completed by Thread[main,5,main]
6 barriers completed by Thread[Thread-0,5,main]
7 barriers completed by Thread[main,5,main]
7 barriers completed by Thread[Thread-1,5,main]
7 barriers completed by Thread[Thread-0,5,main]
8 barriers completed by Thread[Thread-0,5,main]
8 barriers completed by Thread[main,5,main]
8 barriers completed by Thread[Thread-1,5,main]
9 barriers completed by Thread[Thread-1,5,main]
9 barriers completed by Thread[Thread-0,5,main]
9 barriers completed by Thread[main,5,main]
[monali22@cssmpi1h lab1]$
```

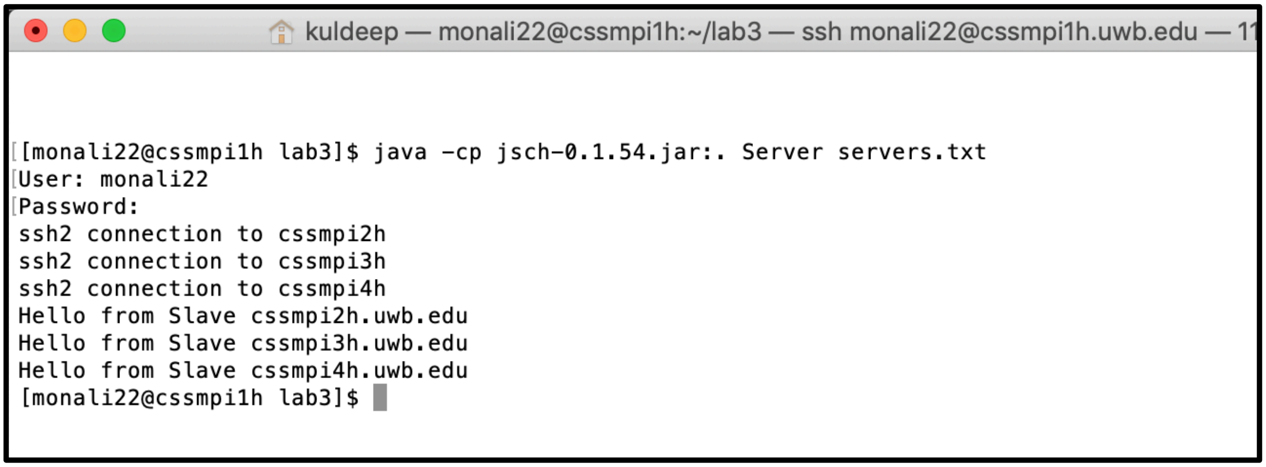
Lab 2 output:


kuldeep — monali22@cssmpi1h:~/lab2 — ssh monali22@cssmpi1h.u...
kuldeep — monali22@cssmpi2h:~/lab2 — ssh monali22@cssmpi2h.uwb.edu — 83...

```
[monali22@cssmpi1h lab2]$ java P2P cssmpi2h hello hi
Usage: java P2P ipAddr port message
[monali22@cssmpi1h lab2]$ java P2P cssmpi2h 12345 hello hi
TCP connection established...
hello from cssmpi2h.uwb.edu
[monali22@cssmpi1h lab2]$
```

```
[monali22@cssmpi2h lab2]$ java P2P cssmpi1h 12345 hello
TCP connection established...
hello from cssmpi1h.uwb.edu
[monali22@cssmpi2h lab2]$
```

Lab 3 output:


kuldeep — monali22@cssmpi1h:~/lab3 — ssh monali22@cssmpi1h.uwb.edu — 11

```
[monali22@cssmpi1h lab3]$ java -cp jsch-0.1.54.jar:. Server servers.txt
User: monali22
Password:
ssh2 connection to cssmpi2h
ssh2 connection to cssmpi3h
ssh2 connection to cssmpi4h
Hello from Slave cssmpi2h.uwb.edu
Hello from Slave cssmpi3h.uwb.edu
Hello from Slave cssmpi4h.uwb.edu
[monali22@cssmpi1h lab3]$ 
```