# CSS533: Program 1


## Shreevatsa Ganapathy Hegde

University of Washington, Bothell

sghegde@uw.edu


Prof. Munehiro Fukuda

*17th April, 2025*

Table of Contents

## Documentation:

The Documentation section involves the implementation details of the OnlineTicTakToe that I have built. This section contains the implementation of the requirements that were given in the program 1 document. The additional features will be discussed in the discussion section.

*OnlineTicTacToe():* This constructor implements a bot which will be used in the single player games. It starts by setting up a connection using connection () of Connection.java so it can send and receive moves, and sets itself as second player as mentioned by the requirements. It also sets up a log file to keep track of what's happening, which helps with debugging. The bot keeps track of which buttons have already been played using a HashSet. Then it enters an infinite loop: if it's the bot's turn, it picks the best available move, sends that move to the other player, and logs what it did. If it's the opponent's turn, the bot waits to receive their move, updates its record, and logs it.

*OnlineTicTacToe(hostname ):* This constructor is used when a human wants to play against the bot. It connects to a remote machine (the hostname) using JSCH over SSH, essentially launching the same OnlineTicTacToe program on that machine, which will act as the bot. The user is prompted to enter their SSH username and password to access the remote host. Once authenticated, it executes the command to start the Java program on the remote server. The local instance acts as the human player ("former"), while the remote instance is the bot ("latter"). Input and output streams are set up to communicate moves between the two, a game window is created for the human player using makeWindow(), and a separate thread (Counterpart) is started to handle incoming moves from the bot.

*OnlineTicTacToe(addr, port ):* This constructor is responsible for setting up a peer-to-peer connection between two players on the same or different machines using a TCP socket. It first tries to create a server socket on the given port — if this succeeds, it means this instance will act as the *server*; if a BindException is caught, it means the port is already in use and a server must be running, so this instance becomes the *client*. In either case, it enters a loop trying to either accept incoming connections (if it's the server) or initiate a connection to the other player (if it's the client). Once a connection is successfully established, it sets up object input and output streams for sending game moves, initializes the game window, and finally starts a background Counterpart thread to handle incoming moves from the other player.

actionPerformed( ActionEvent event ): This method reacts when a player clicks a button. If it's not their turn, it does nothing. Otherwise, it marks the chosen cell, sends the move to the opponent, and checks for a win or draw. If the game ends, it shows appropriate message.

*Counterpart Class:* This thread runs in the background and handles the opponent's actions during the game. It waits for input from the other player, marks their move on the board, and checks for a win or draw. If the game ends, it shows the result, and disables the board. It also flips the turn back to the player after each valid move.

## Source Code:

To keep the documentation concise, only the source code for the methods mentioned above are included here. The full implementation can be found in the OnlineTicTacToe.java file, which will be submitted along with this report. The Source formatting is kept to keep the document concise. I have given comments whenever required. Please do contact me if it needs further explaination.

OnlineTicTacToe():

```
public OnlineTicTacToe( ) throws IOException {
    // receive an ssh2 connection from a user-local master server.
    Connection connection = new Connection();
    input = connection.in;
    output = connection.out;

    // 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( "logs.txt" ) );
    logs.println( "Autoplay: got started." );
    logs.flush( );

    myMark = "X";   // auto player is always the 2nd.
    yourMark = "O";

    isBot = true; // this is the auto player

    // the main body of auto play.
    // IMPLEMENT BY YOURSELF
    myTurn[0] = false; // this is the auto player
    Set<Integer> set = new HashSet<>(); // to keep track of marked buttons
    while (true) {
            if (myTurn[0]) {
                int button =  getBestMove(set); // get the best move
                set.add(button);
                output.writeObject(button); // send the button id to the counterpart
                output.flush();
                logs.println("Autoplay marked" + button);
                logs.flush();
            }

            if (!myTurn[0]) {
                try {
                    int yourButton = (int) input.readObject(); // read the button id
from the counterpart
                    set.add(yourButton); // add the button to the set
                    logs.println("Oppnent played " + yourButton);  // log the move
```

```
                    logs.flush();
                }
                catch (ClassNotFoundException e) {
                    error(e);
                }
            }
            myTurn[0] = !myTurn[0]; //Change turn
        }

    }
```

*OnlineTicTacToe(hostname ):*

```
public OnlineTicTacToe( String hostname ) {
        final int JschPort = 22;      // Jsch IP port

        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
        // Read the username and password from the console
        System.out.print("Username: ");
        username = keyboard.nextLine();
        Console console = System.console();
        password = new String(console.readPassword("Password: "));

        // The command to be executed on the remote server
        String cur_dir = System.getProperty("user.dir");
        String command = "java -cp " + cur_dir + "/jsch-0.1.54.jar:" + cur_dir + "
OnlineTicTacToe";

        // establish an ssh2 connection to ip and run
        // Server there.
        Connection connection = new Connection( username, password,
                        hostname, command );
        System.out.println("Connection established with " + hostname);

        // 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();
        counterpart.start();
    }
```

*OnlineTicTacToe(addr, port ):*

```
 public OnlineTicTacToe( InetAddress addr, int port ) {
        // set up a TCP connection with my counterpart
        // IMPLEMENT BY YOURSELF
        System.out.println("Trying to Connect to " + addr + ":" + port);
        ServerSocket server = null;
        boolean isServer = false;
        boolean isBusy = false;
        // try to create a server socket
        try {
            server = new ServerSocket( port );
            server.setSoTimeout(INTERVAL);
        }catch ( BindException e){
            //BinedException is thrown when the port is already in use
            // meaning the server is already running
            isBusy = true;
        } catch ( Exception e ) {
            error( e );
        }

        Socket client = null;
        while ( true ) {
            //check for localhost
            if(addr.getHostName().equals("localhost")){
                if(!isBusy){
                    // if the port is not busy meaning the server is not running, you
are the server
                    // wait for a connection
                    try {
                        client = server.accept();
                    } catch (SocketTimeoutException e) {
                        // Timeout, continue waiting
                    } catch (IOException e) {
                        error(e);
                    }
                    // Check if a connection was established. If so, leave the loop
                    if (client != null) {
                    isServer = true;
```

```
                    break;
                    }
                }else{
                    // if the port is busy, Server is running, you are the client
                    // try to connect to the server
                    try {
                        client = new Socket( addr, port );
                    } catch (IOException e) {
                        // Connection failed, continue waiting
                    }
                    if (client != null) {
                    break;
                    }
                }

            } else {
                if (!isBusy) {
                    // if the port is not busy meaning the server is not running, you
are the server
                    // wait for a connection
                    try {
                        client = server.accept();
                    } catch (SocketTimeoutException e) {
                        // Timeout, continue waiting
                    } catch (IOException e) {
                        error(e);
                    }
                    // Check if a connection was established. If so, leave the loop
                    if (client != null) {
                        isServer = true;
                        break;
                    }
                }
                // if the port is busy, Server is running, you are the client
                // try to connect to the server
                try {
                    client = new Socket( addr, port );
                } catch (IOException e) {
                    // Connection failed, continue waiting
                }
                if (client != null) {
                    break;
                }

            }
```

```
        }
        try{
            System.out.println("Connected to " + client.getInetAddress() + ":" +
client.getPort());
            // set up a window
            makeWindow( !isServer );
            // set up input and output streams
            output = new ObjectOutputStream( client.getOutputStream() );
            input = new ObjectInputStream( client.getInputStream() );
        } catch (Exception e){
            error(e);
        }
        // start my counterpart thread
        Counterpart counterpart = new Counterpart( );
        counterpart.start();
    }
```

actionPerformed( ActionEvent event ):

```
public void actionPerformed( ActionEvent event ) {
    // IMPLEMENT BY YOURSELF
        if(!myTurn[0]){
            return;
        }
        int button = whichButtonClicked(event); // check which button was clicked
        if(button == -1) return;
        if(markButton(button, myMark)){
            try {
                output.writeObject(button); // send the button id to the counterpart
                output.flush();
                myTurn[0] = false; // change turn
            }catch(IOException e){
                error(e);
            }
            // Check if the current player has won
            if (checkWin(myMark)) {
                showWon(myMark);
                window.setEnabled(false);
                restart();  // To restrat the game
                // System.exit(0);
            }
            // Check if the game is a draw
            if (checkDraw()) {
                JOptionPane.showMessageDialog(null, "It's a draw!");
```

```
                window.setEnabled(false);
                restart(); // To restrat the game
                // System.exit(0);
                return;
            }
        }
    }
```

*Counterpart Class:*

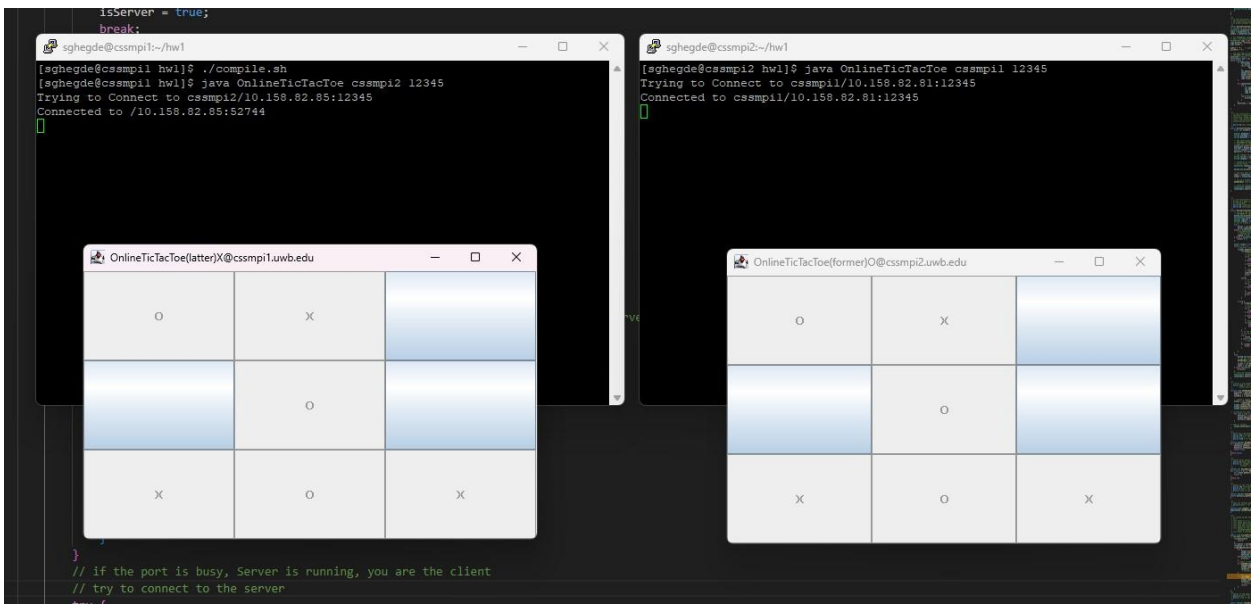
```
private class Counterpart extends Thread {

    /**
     * Is the body of the Counterpart thread.
     */
        @Override
        public void run( ) {
        // IMPLEMENT BY YOURSELF
        try{
            while(true){
                Object obj = input.readObject();
                if(obj instanceof Integer){
                    int button = (Integer) obj;
                    markButton(button, yourMark);

                 // Check if the current player has won
                    if(checkWin(yourMark)){
                        showWon(yourMark);
                        window.setEnabled(false);
                        restart(); // To restrat the game
                        // System.exit(0);
                        break;
                    }
                    // Check if the game is a draw
                    if(checkDraw()){
                        JOptionPane.showMessageDialog(null, "It's a draw!");
                        window.setEnabled(false);
                        restart(); // To restrat the game
                        // System.exit(0);
                        break;
                    }
                    // Change turn
                    myTurn[0] = true;
                }
            }
```

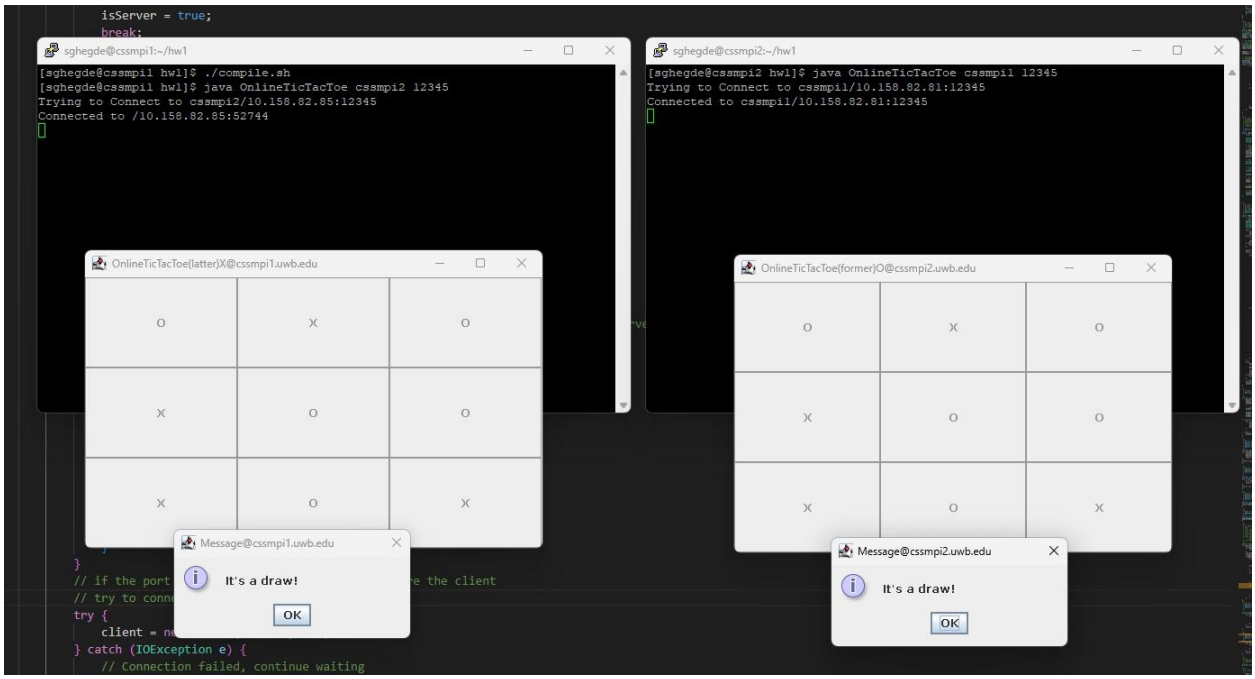
```
        }catch(Exception e){
            error(e);
        }
        }
    }
```

## Execution Outputs:

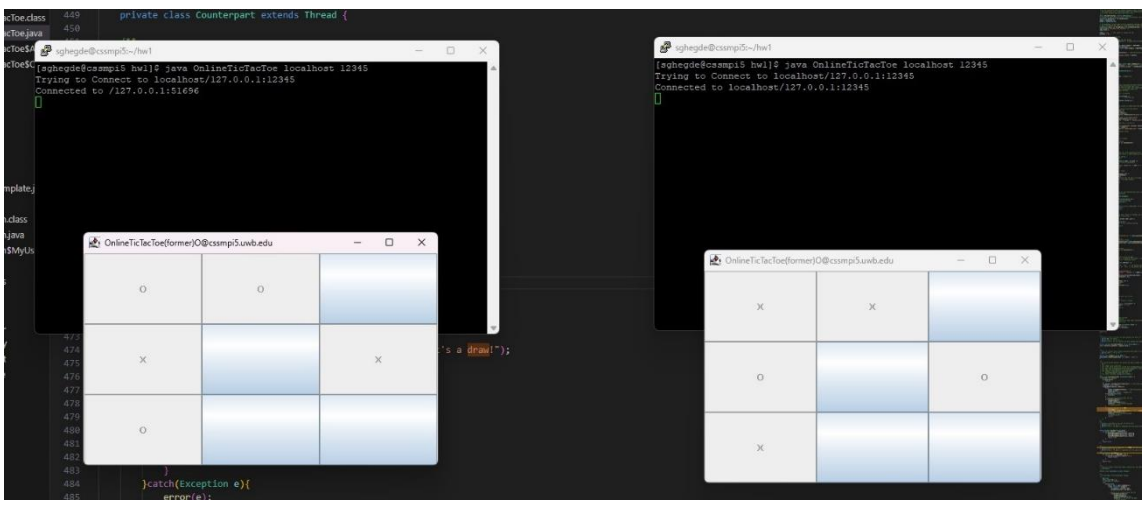
Three scenarios were given to test with the implementation. Below is the outputs for each.

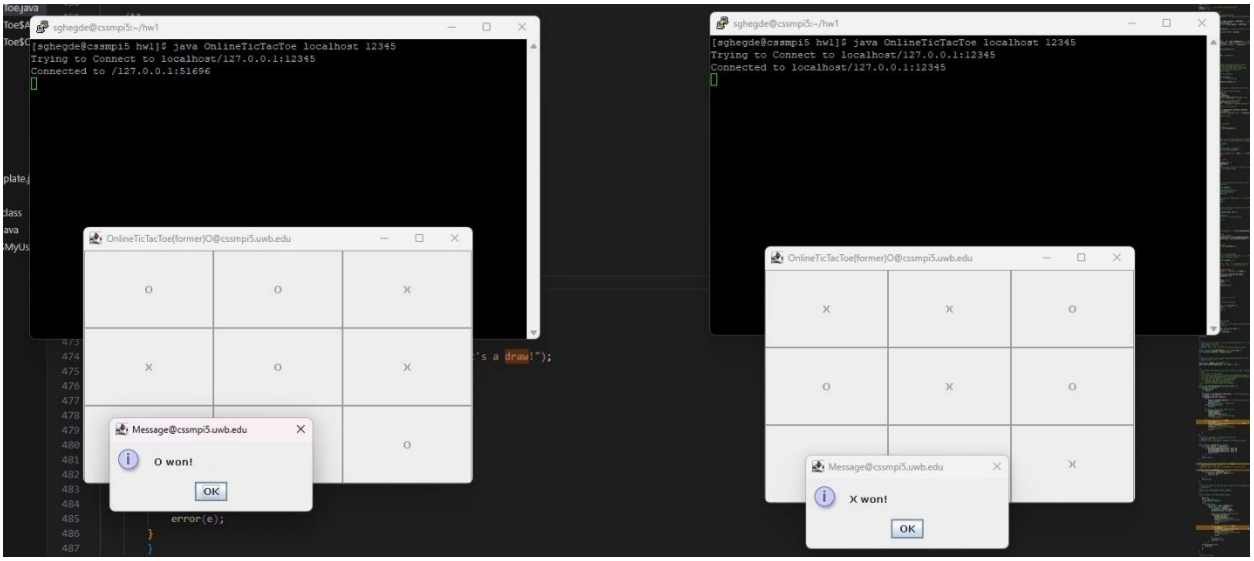
a. Running a two-user interactive game over two different machines



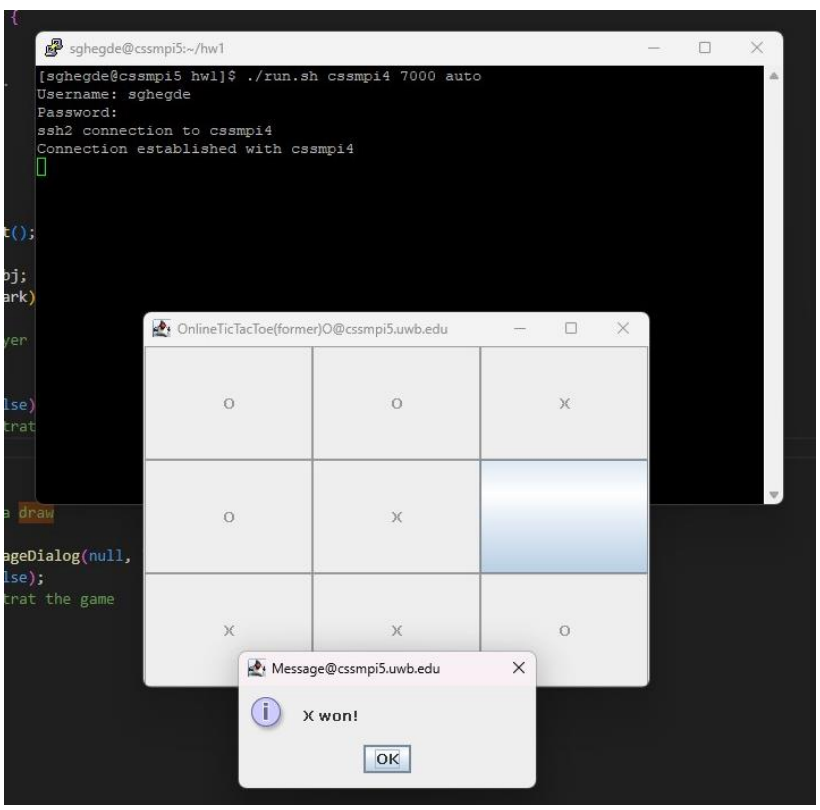



b. Running a two-user interactive game at one machine






c. Running a single-user automated game over two different machines



## Discussions:

Additional Features:

1. Added a new draw or tie feature (which was not there before).
2. Made the automation bot a little bit intelligent. Tried to make the bot choose choices that as a human we would go for such as prioritizing centre slot or corner when available.
3. Tried to implement the reset feature. This is not fully implemented yet. This gives an error at the end but this will be in my next improvements.

To make the above improvements I added couple of methods to the file.

For the Tie or draw feature, I added a checkDraw() method.

checkDraw(): The method checks if the game has ended in a draw. It goes through all the Tic Tac Toe buttons, and if it finds any button that hasn't been clicked (i.e., still empty), it returns false—meaning the game is still ongoing. If all buttons are filled and no winner has been declared, it returns true, indicating the game is a draw.

For the implementation of intelligent bot, the two following methods were added.

*getBestMove():* The getBestMove method helps the bot play smart by following a priority-based strategy. It first looks for a winning move, then checks if it needs to block the opponent. If neither is possible, it tries to take the center, then a corner, and finally a side. This ensures the bot makes strategic decisions instead of picking moves randomly.

*findWinningMove():* This method helps the bot spot a winning move or block the opponent. It looks at a possible winning pattern and checks how many spots already have the given mark (X or O). If two are filled and one is still empty, it returns that empty spot so the bot can either finish the game or stop the opponent from winning. If there's no such opportunity in that pattern, it simply returns -1.

For the reset functionality, one restart() method is added and changes to some methods such as actionPerformed and Counterpart.run which are directly responsible in deciding the end of the game.

*Restart():* This method handles restarting the game after it ends. It first asks the player if they want to play again. If it's a multiplayer game, it also waits for the opponent's response. If either player chooses not to continue, the game exits. If both agree to restart, it clears the board and re-enables the buttons based on whose turn it is.

Limitations & improvements.

The current restart() method lacks synchronization between the two players. Although it waits for both sides to confirm a restart, the player who clicks first often fails to update the board properly. Debugging shows that only the second player's board resets correctly. Attempts to implement restart logic exclusively for the bot introduced additional bugs, such as connections being unexpectedly closed after a new game is initialized. These issues highlight some current limitations in the restart logic and areas for future improvement.

# Lab 1

EXPLORER

SGHEGDE [SSH: CSSMPI5....
- .cache
- .dotnet
- .java
- .vscode-server
- hw1
- lab1
  - .BarrierThreadIncom...
  - BarrierThread.class
  - BarrierThreadIncom...
  - BarrierThreadIncom...
- lab2
  - classs
  - P2P.class
  - P2P.java
- lab3
  - #Server_template.ja...
  - compile.sh
  - Connection.class
  - Connection.java
  - Connection$MyUser...
  - run.sh
  - Server.class
  - Server.java
  - servers.txt
  - servers.txt~
- .bash_history
- .bash_logout
- .bash_profile
- .bashrc
- .emacs
- .viminfo
- .vimrc
- .wget-hsts
- .Xauthority
- logs.txt

OnlineTicTacToe.java | *BarrierThreadIncomplete.java*

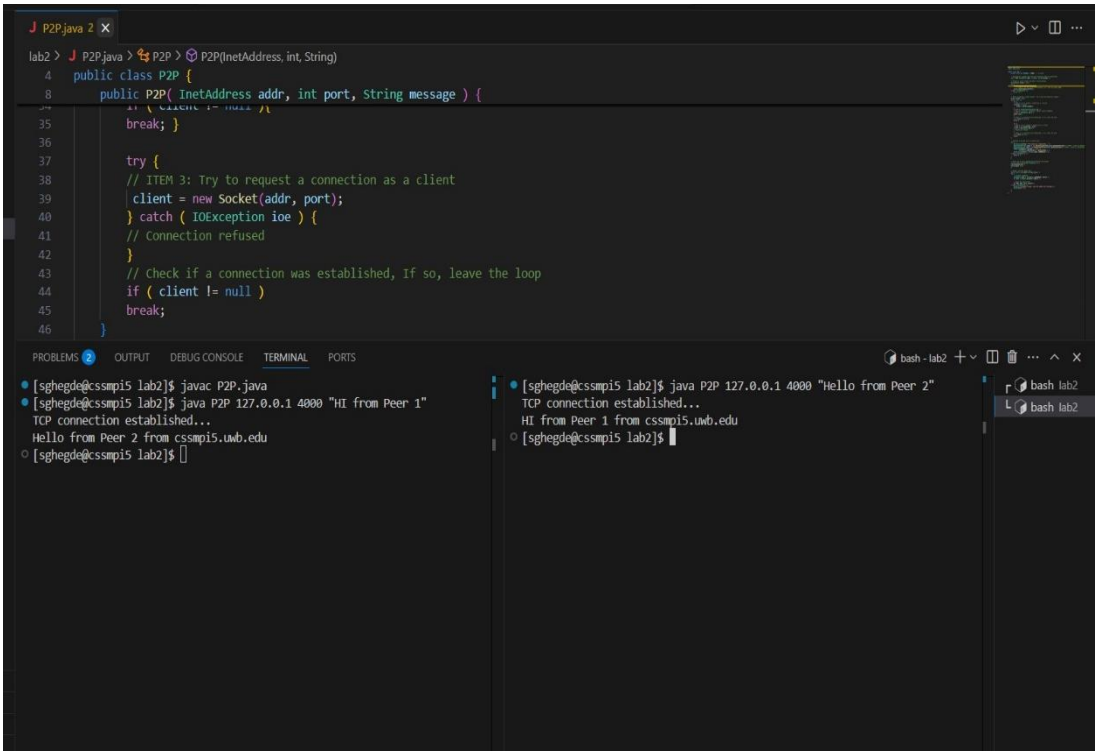
lab1 > BarrierThreadIncomplete.java

```java
class BarrierThreadIncomplete implements Runnable {
    }

    // this is what you implement
    private void barrier( ) {
    //synchronized( sync ) {
        // increment sync[0], because I reached the barrier
        // if sync[0] does not reach N, #threads, there must be someone else who has not cal
        //   let's wait
        // else, all threads called barrier( ) and I am the last
        //   let's wake them all
        //   zero-initialize sync[0] for the next barrier
    //
    //}
    synchronized(this) {
        this.sync[0]+=1;
        while(this.sync[0] != N) {
            try{
                this.wait();
            }catch(InterruptedException e) {}
        }

        this.notifyAll();
        this.sync[0] = 0;
        }
    }

    public static void main( String args[] ) {
    // java BarrierThread #Threads iterations
    int[] sync = new int[1]; // used to count the number of threads that called barrier so t
```

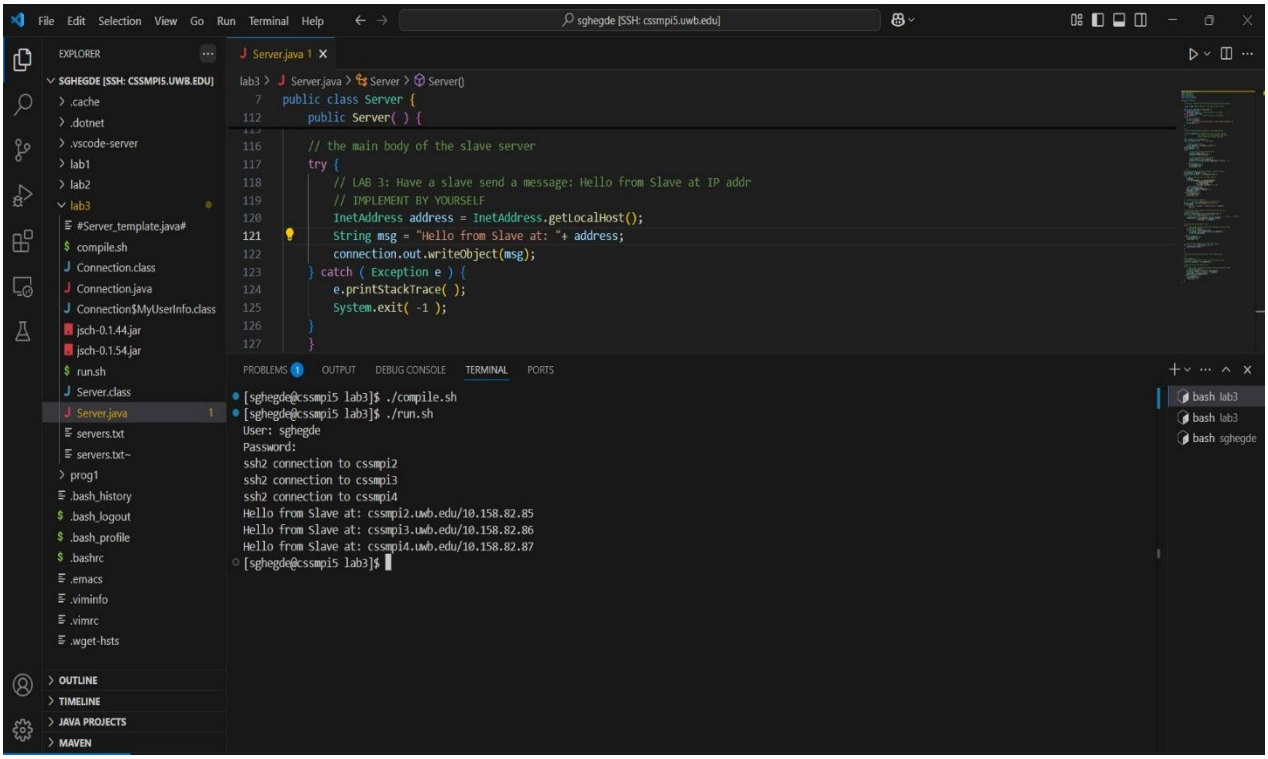
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[sghegde@cssmpi5 lab1]$ java BarrierThreadIncomplete 3 2
0 barriers completed by Thread[Thread-1,5,main]
0 barriers completed by Thread[Thread-0,5,main]
0 barriers completed by Thread[main,5,main]
1 barriers completed by Thread[main,5,main]
1 barriers completed by Thread[Thread-0,5,main]
1 barriers completed by Thread[Thread-1,5,main]
[sghegde@cssmpi5 lab1]$
```

# Lab -2


P2P.java 2
lab2 > P2P.java > P2P > P2P(InetAddress, int, String)
public class P2P {
public P2P( InetAddress addr, int port, String message ) {
break; }
try {
// ITEM 3: Try to request a connection as a client
client = new Socket(addr, port);
} catch ( IOException ioe ) {
// Connection refused
}
// Check if a connection was established, If so, leave the loop
if ( client != null )
break;
}
PROBLEMS 2
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
bash - lab2
[sghegde@cssmpi5 lab2]$ javac P2P.java
[sghegde@cssmpi5 lab2]$ java P2P 127.0.0.1 4000 "HI from Peer 1"
TCP connection established...
Hello from Peer 2 from cssmpi5.uwb.edu
[sghegde@cssmpi5 lab2]$
[sghegde@cssmpi5 lab2]$ java P2P 127.0.0.1 4000 "Hello from Peer 2"
TCP connection established...
HI from Peer 1 from cssmpi5.uwb.edu
[sghegde@cssmpi5 lab2]$
bash lab2
bash lab2

# Lab-3


File Edit Selection View Go Run Terminal Help
sghegde [SSH: cssmpi5.uwb.edu]
EXPLORER
SGHEGDE [SSH: CSSMPI5.UWB.EDU]
.cache
.dotnet
.vscode-server
lab1
lab2
lab3
#Server_template.java#
compile.sh
Connection.class
Connection.java
Connection$MyUserInfo.class
jsch-0.1.44.jar
jsch-0.1.54.jar
run.sh
Server.class
Server.java
servers.txt
servers.txt~
prog1
.bash_history
.bash_logout
.bash_profile
.bashrc
.emacs
.viminfo
.vimrc
.wget-hsts
OUTLINE
TIMELINE
JAVA PROJECTS
MAVEN
Server.java 1
lab3 > Server.java > Server > Server()
public class Server {
public Server( ) {
// the main body of the slave server
try {
// LAB 3: Have a slave send a message: Hello from Slave at IP addr
// IMPLEMENT BY YOURSELF
InetAddress address = InetAddress.getLocalHost();
String msg = "Hello from Slave at: "+ address;
connection.out.writeObject(msg);
} catch ( Exception e ) {
e.printStackTrace( );
System.exit( -1 );
}
}
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS
[sghegde@cssmpi5 lab3]$ ./compile.sh
[sghegde@cssmpi5 lab3]$ ./run.sh
User: sghegde
Password:
ssh2 connection to cssmpi2
ssh2 connection to cssmpi3
ssh2 connection to cssmpi4
Hello from Slave at: cssmpi2.uwb.edu/10.158.82.85
Hello from Slave at: cssmpi3.uwb.edu/10.158.82.86
Hello from Slave at: cssmpi4.uwb.edu/10.158.82.87
[sghegde@cssmpi5 lab3]$
bash lab3
bash lab3
bash sghegde