
# CS 421- Computer Networks

## Programming Assignment 2

26.05.2023

**Instructor:** Ezhan Karaşan

**Section:** 01

Zeynep Selcen Öztunç - 21902941

## The Structure of My Code

In this assignment I have implemented a client-server Tic-Tac-Toe in Java. I have classes named TicTacToeServer, TicTacToeClient, TicTacToe, TicTacToeGame, Client and Exceptions. Let me first explain the general structure of TicTacToeClient. The TicTacToeClient class represents the client side of a client-server TicTacToe game. The class establishes a connection to the TicTacToe server by creating a Socket object and specifying the server's hostname and port number.It sets up input and output streams to communicate with the server. The OutputStream object output is used to send messages to the server, and the InputStream object input is used to receive messages from the server. The class creates a separate thread named read to continuously read input from the server. Inside the thread, it reads the data until it encounters the ASCII value 38 (corresponding to the '&' character). Another thread named write is created to continuously read user input from the console. Inside the thread, the user's input is then sent to the server by writing it to the output stream. The message is converted to bytes and appended with '&' before being sent.

```
15
16      //get the port number from the terminal
17      //int port = 12345;
18      int port= Integer.parseInt(args[0]);
19      Socket socket = new Socket(host:"localhost", port);
20      System.out.println(x:"Connected to the server.");
21
22      //init the input and outpur stream
23      OutputStream output = socket.getOutputStream();
24      InputStream input = socket.getInputStream();
25
26      //create a thread to read input streams
27      Thread read = new Thread(() -> {
28          while (true) {
29              try {
30                  if (input.available() > 0) {
31                      int data;
32                      StringBuilder msg = new StringBuilder();
33                      while ((data = input.read()) != 38) {
34                          msg.append((char) data);
35                      }
36                      System.out.println(msg.toString());
37                  }
38              } catch (IOException e) {
39                  e.printStackTrace();
40              }
41          }
42      });
43
44      read.start();
45
46      //create a thread to read the output stream
47      Thread write = new Thread(() -> {
48          Scanner scan = new Scanner(System.in);
49          while (true) {
50              String msg = scan.nextLine();
51              try {
52                  output.write((msg + "&").getBytes());
53                  output.flush();
54              } catch (IOException e) {
55                  e.printStackTrace();
56              }
57          }
58      });
59
60      write.start();
61
62  }
```

**TicTacToeServer** class represents the server side of a client-server TicTacToe game. The class listens for incoming connections from clients on a specified port number. It creates a ServerSocket object to listen for client connections on the specified port. The server accepts incoming client connections using the accept() method of the ServerSocket class. It creates a

Client object representing the player and assigns it to either player 1 or player 2 based on the order of connection. Once both players are connected, the server creates a GameEngine object and passes the player instances to it. The server starts the game by calling the run() method of the GameEngine object. and the server repeats this process indefinitely, accepting new connections and starting new games as clients connect.

```java
public class TicTacToeServer {
    Run | Debug
    public static void main(String[] args) throws IOException {

        //give an error if the command line arguments are wrong
        if (args.length != 1) {
            System.out.println(x:"You should enter the following way: java TicTacToeServer <port>");
            return;
        }

        //get the port number from the terminal
        //int port = 12345;
        int port= Integer.parseInt(args[0]);
        ServerSocket serverSocket = new ServerSocket(port);
        Socket playerSocket;

        while (true) {
            //create the first player
            playerSocket = serverSocket.accept();
            Client player1 = new Client(playerSocket);
            System.out.println(x:"A client is connected, and it is assigned with the symbol X and ID=0.");

            //create the second player
            playerSocket = serverSocket.accept();
            Client player2 = new Client(playerSocket);
            System.out.println(x:"A client is connected, and it is assigned with the symbol O and ID=1.");

            //start the game
            TicTacToeGame newGame = new TicTacToeGame(player1, player2);
            System.out.println(x:"The game is started.");
            newGame.run();
        }
    }
}
```

The **TicTacToeGame** class represents a game instance of Tic-Tac-Toe between two clients (player1 and player2). It implements the Runnable interface, which allows it to be executed in a separate thread. The class does the following:
It initializes the two clients (player1 and player2) and creates a new instance of the TicTacToe game.
The playGame() method is responsible for handling the game logic. It takes the current player (c1), the opponent player (c2), and the mark (symbol) for the current player's moves. It interacts with the clients to make moves, check for game outcomes (such as winning or drawing), and send game state information to the clients.

```
private boolean playGame(Client c1, Client c2, String mark) {
    String msg;
    try {
        boolean marked = false;
        while (!marked) {
            try {
                c1.writeToClient(msg:"Please enter your move 1-9: ");
                msg = c1.readFromClient();    //read client1 next move
                game.makeMove(msg, mark); //send client1 move to the game program
                marked = true;     // if the given move is marked successfully then exit
            } catch (PositionAlreadyMarkedException ge) {

                c1.writeToClient(msg:"This is an illegal move. Please change your move!");
            }
        }

        //send current state to the players
        c1.writeToClient(game.arrayToString());
        c2.writeToClient(game.arrayToString());

        c1.writeToClient(msg:"Turn information: Player 1's turn");
        c2.writeToClient(msg:"Turn information: Your turn");
    } catch (PlayerWonException pwe) {
        c1.writeToClient(msg:"\n.You have won the game \n" );
        c2.writeToClient(msg:"\nYou have lost the game \n");
        c1.writeToClient(game.arrayToString());
        c2.writeToClient(game.arrayToString());
        return true;
    } catch (GameDrawException gde) {
        c1.writeToClient(msg:"It is a draw!");
        c2.writeToClient(msg:"It is a draw!");
        c1.writeToClient(game.arrayToString());
        c2.writeToClient(game.arrayToString());
        return true;
    }
    return false;
}
```

The run() method, which is executed when the TicTacToeGame is run as a separate thread, handles the main game loop. It starts by sending the initial state of the board to both players. Then, it iterates until the game is over (a player wins or a draw occurs). In each iteration, it determines the current player and their symbol, and calls playGame() to handle their turn. After the game ends, it sends the final state of the board to player1 and closes the connections for both players.

```
@Override
public void run() {
    // Display the initial state of the board
    String initialBoardState = "State of the board\n\n" + game.arrayToString();
    player1.writeToClient(initialBoardState);
    player2.writeToClient(initialBoardState);

    boolean finish = false;
    boolean player1Turn = true;
    String player1Symbol = "X";
    String player2Symbol = "O";

    while (!finish) {
        // Current player's turn
        Client currentPlayer = player1Turn ? player1 : player2;
        Client otherPlayer = player1Turn ? player2 : player1;
        String currentPlayerSymbol = player1Turn ? player1Symbol : player2Symbol;

        finish = playGame(currentPlayer, otherPlayer, currentPlayerSymbol);
        player1Turn = !player1Turn;
    }

    // Display the final state of the board
    String finalBoardState = "State of the board\n" + game.arrayToString();
    player1.writeToClient(finalBoardState);

    // Close connections
    player1.closeConnections();
    player2.closeConnections();

}
```

Overall, the **TicTacToeGame** class encapsulates the game logic and communication between the server and the clients for a Tic-Tac-Toe game. It allows multiple games to be played concurrently by creating separate instances of this class for each game.

The **TicTacToe** class represents the game logic for a Tic-Tac-Toe game. The class has an instance variable board, which is an array of integers representing the game board. The

constructor initializes the count variable to 0 and creates a new array for the game board. The checkForWin() method checks for winning combinations on the game board and throws a PlayerWonException if a player has won:

```
private void checkForWin(int sum) throws PlayerWonException {
    int[][] winningCombinations = {
        {0, 1, 2}, {0, 3, 6}, {2, 5, 8}, {6, 7, 8},
        {0, 4, 8}, {1, 4, 7}, {2, 4, 6}, {3, 4, 5}
    };

    for (int[] combination : winningCombinations) {
        if (board[combination[0]] + board[combination[1]] + board[combination[2]] == sum) {
            String positions = Arrays.toString(combination)
                    .replaceAll(regex:"\\[|\\]|\\s", replacement:"")
                    .replace(target:",", replacement:", ");
            throw new PlayerWonException(positions);
        }
    }
}
```

The shouldGameContinue() method determines if the game should continue based on the current state of the game board. It throws a GameDrawException if the game is a draw. The mark() method marks a position on the game board with the given mark index. It throws a PositionAlreadyMarkedException if the position is already marked. The makeMove() method handles a player's move by parsing the position and mark from the input. It calls the mark() method to mark the position and the shouldGameContinue() method to check if the game should continue. The printArray() method converts the game board array into a string for display.Overall, this class provides methods to handle player moves, check for wins or draws, and represent the game board as a string.

The **Client** class represents a client connection in a client-server communication setup. The class has instance variables socket, reader, and writer to handle the communication with the client.The constructor Client(Socket socket) initializes the client object with the provided socket, and sets up the input and output streams for communication. The writeToClient(String msg) method writes a message to the client by converting it to bytes and sending it through the output stream. It appends a & character at the end of the message to mark the end of the message.The readFromClient() method reads data from the client by continuously checking for available data in the input stream. It reads the data until it encounters the & character, indicating the end of the message. The method returns the received message as a string. The closeConnections() method is responsible for closing the socket, input stream, and output stream to properly terminate the client connection. So this class provides functionality to manage the lifecycle of the client connection by opening and closing the necessary streams and socket.

Finally the **Exceptions** class defines custom exception classes related to the Tic-Tac-Toe game. For example, the Exceptions class is the base class for all game-related exceptions. It extends the Exceptions class and serves as a general category for game-specific exceptions. PositionAlreadyMarkedException represents an exception that occurs when a player tries to mark a position on the game board that is already occupied. It indicates an invalid move by a player. GameDrawException: represents an exception that occurs when the game ends in a draw. It indicates that all positions on the game board are filled, and no player has won. PlayerWonException represents an exception that occurs when a player wins the game. It includes a pos field that specifies the winning positions on the game board. It is used to indicate that a player has achieved a winning combination.