1603930

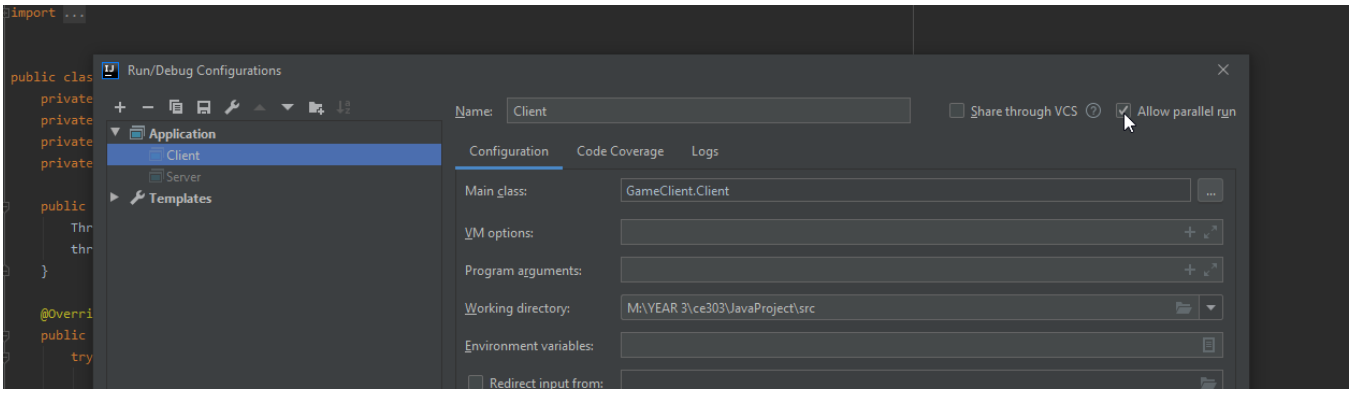
CE303-6

Report

Advanced Programming

Running the Server and Clients

The Java program was developed in such a way that both Server and Client can be ran through one project (However, Clients can be ran separately from Server package). To run more than once instance of client through IntelliJ the user needs to enable “Allow parallel run”



Implemented Functionality

Function	C#	Java
Client establishes a connection with the server	Yes	Yes
Client is assigned a unique ID when joining the game	Yes	Yes
Client displays up-to-date information about the game state	Partial – sends information successfully to clients that do not have ball.	Yes - refreshes when user enters 0 if player has ball, otherwise displays every second.
Client allows passing the ball to another player	No	Yes
Server manages multiple client connections	Yes	Yes
Server accepts connections during the game	Yes	Yes
Server correctly handles clients leaving the game	Partial – Console outputs are incorrect but clients leaving are removed from databases.	Yes
Client is compatible with the server in the other language	No	No
Additional tasks:		
Client GUI	No	No
Server GUI	No	No
Server restarts are correctly implemented	No	No
Unit tests	No	No

Protocol

Client connects to socket.

Client opens data input stream and data output stream.

Client listens to input stream and expects data string that relays information form Server consisting of the Client ID, Connected Players and which Client has the ball (this message is passed to Client at every specified time period).

Client receives Boolean data that determines if Client has the ball.

Client (if has ball – determined by Server) sends integer to the Server of a player number that he/she wishes to pass the ball to.

Client socket is closed upon termination.

Program Architecture

My intent was to program both the Client and Server at the same time, so for this reason I implemented three packages. Utilities, GameClient and GameServer. Both use the Utilities package which contains the class Constants.

This made my code cleaner and decreased the amount of repetitions as Constants, for example, contains the port, IP address and delay (the delay controls the data output from the server to all clients).

The GameServer package consists of a Ball.java class which is there to mainly store methods related to game logic.

Requirements

Drawing from the assignment document, I came up with these requirements that I aimed to fulfill.

Application

1. Socket-based client-server system.
2. Each client application is a player.
3. Clients can connect/disconnect at any given time.
4. Multi-threading for every client connection.

Game Rules

5. If a players are present, one must have control of the ball.
6. Player with ball has power to grant someone the ball.
7. New players get assigned unique ID that cannot be reused.
8. If ball owner leaves and if players are present, ball must be handed to someone.
9. If no players present, server waits for connection and assign first player the ball.

Client User Interface

10. Each client displays Client ID.
11. Each client display information about connected players.
12. Each client displays ball owner.
13. Server needs to display players joining.
14. Server needs to display players in the game (after joining and/or leaving)
15. Server needs to display players leaving.
16. Server needs to display a message when ball gets assigned automatically (first player or last player).
17. Server needs to display both clients of ball passing interaction.

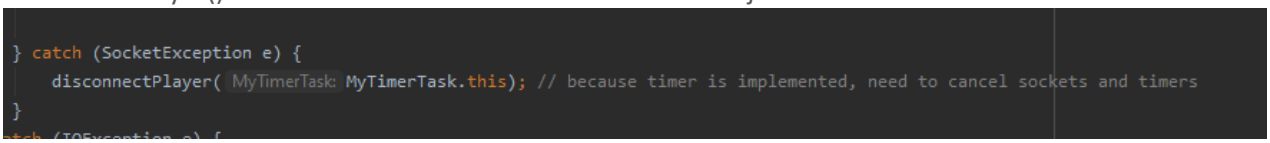
Further Requirements

18. Validation for gameplay
19. Additional tasks if possible.

Product compared to Requirements

For simple tasks, I briefly explain if requirements have been met.

1. This program connects through sockets in a client-server system.
2. The client application can be completely standalone (if utilities.Constants is implemented into the actual code or even a standalone Client project).
3. Due to nature of sockets, clients can connect and disconnect at any time. Clients can only exit by terminating the process which makes the use of the program swift. This is done by implementing a disconnectPlayer() method in the catch block in ClientHandler.java



   This catches the termination and when it is triggered this method is executed which allows for a desired result.
4. This one done by implementing two classes for the Server. One called Server.java which generally initializes the server (sets up databases, looks for socket connection etc…) and the other called ClientHandler.java. ClientHandler.java extends Thread and has a constructor which contains the stream for both data input and out, socket and ClientID. This allows the stream objects to be pass into ClientHandler.java, where tasks within the actual thread will be processed.

```
// obtain input/output streams
DataInputStream dis = new DataInputStream(socket.getInputStream());
DataOutput dos = new DataOutputStream(socket.getOutputStream());
Server.dos = (DataOutputStream) dos;

// creating new thread for object
Thread thread = new ClientHandler(dis, dos, socket, clientID);
// starting thread
thread.start();
```

```
public ClientHandler(DataInputStream dis, DataOutput dos, Socket socket, int clientID) {
    this.dis = dis;
    this.dos = dos;
    this.socket = socket;
    this.clientID = clientID;
}
```

5. To help with dealing with game logic I implemented one database at first (HashMap<Socket, ClientID>) and then another (TreeSet<ClientID, Bool hasBall>).

```
public class Server {
    public static HashMap<Socket, Integer> connectedClients = new HashMap<>();
    public static TreeMap<Integer, Boolean> gameState = new TreeMap<>();     // Integer clientID, Boolean hasBall
```

The TreeSet was implemented as to be able to get last entries which are then used to find a player to assign the ball to.

```
public void disconnectBall() { // when player with ball leaves, give a player the ball
    if (Server.gameState.containsValue(false) && Server.gameState.size() >= 1) { // if no one has ball and more than 1 player
        Map.Entry<Integer, Boolean> lastEntry = Server.gameState.lastEntry(); // finds last entry
        System.out.println("Automatically passing ball to ClientID: " + lastEntry.getKey());
        Server.gameState.replace(lastEntry.getKey(), true); // uses last entry key and replaces its bool value with true
```

6. This was simply implemented by changing the value of selected key (chosen Client ID by current ball holder) in the gameState TreeMap to true.
7. A simple solution, each time a player joins (socket connects) increment int ClientID and assign. This way the ID is always unique, especially when old ID's are not reused.
8. Mention above, program checks to see if no one has the ball and if there is more than one player active. It then gets the last entry (last because first are always new connections) and assigns that player the ball.

```
public static void checkIfFirstPlayer(int clientID) {
    if (Server.gameState.isEmpty()) { // if no players
        Server.gameState.put(clientID, true); // first player gets ball
        System.out.println("First player! Ball is handed to ClientID: " + clientID);
    } else {
        Server.gameState.put(clientID, false); // else we just put in other players as false
    }
}
```

9. This method in Server.java checks if the game field is empty, if it isn't then it assigns the first Client the ball. Otherwise, adds player to database but marks them as not having the ball.

10. – 12.

```
try {
    ClientHandler.this.dos.writeUTF( s: "You are Player: " + returnClientID() + "\n" + "CONNECTED PLAYERS: \n" + Server.stringPlayers() + "\n" + "Player: " + ball.whoHasBall() + " has the ball!\n");
    ClientHandler.this.dos.writeBoolean(ball.doesClientHaveBall(ClientHandler.this.clientID));
```

This long line of code prints out all the necessary information in one string. The client receives this code as a string and as a result is printed whilst data input stream is available.

```
    @Override
    public void run() {
        timer.schedule(new MyTimerTask(),  delay: 0, Constants.delay);

    }

    class MyTimerTask extends TimerTask {

        @Override
        public void run() {
            int received;
            // System.out.println("        Active threads: "+ Thread.activeCount());
            try {
                try {
                    ClientHandler.this.dos.writeUTF( s: "You are Player: " + returnClientID() + "\n" + "CONNECTED PLA
                    ClientHandler.this.dos.writeBoolean(ball.doesClientHaveBall(ClientHandler.this.clientID));
```

A timer task is implemented here with a delay of 1000 (stated in constants). This, when implemented in the thread ClientHandler, lets the long string message be sent to each Client every second as to refresh the player list.

```
while (dis.available() > 0) {
    System.out.println(dis.readUTF());
    hasBall = dis.readBoolean();
```

13. – 17.

```
public static String stringPlayers() {
    String string = "";
    for (int value : Server.connectedClients.values()) {
        string = string + "Client ID: " + value + "\n";
    }
    return string;
}
```

This method is used in the picture above, it simply iterates in the connectedClients hashmap, gets its value and adds it to a string.

```
    checkIfFirstPlayer(clientID); // checks if first player then adds to treemap

    System.out.println("ClientID: " + clientID + ", " + " Connected successfully.");
    System.out.println();
    System.out.println("CONNECTED PLAYERS: ");
    System.out.println(stringPlayers());
```

When a client connects…

```
public void disconnectPlayer(TimerTask MyTimerTask) throws IOException {
    System.out.println("Player: " + this.clientID + ", " + " Connection closed.");
    Server.connectedClients.remove(this.socket, this.clientID); // removes player

    // ball
    ball.removeBall(this.clientID);
    Server.gameState.remove(this.clientID); // removes player from state
    ball.disconnectBall();

    System.out.println();
    System.out.println("CONNECTED PLAYERS: ");
    if (Server.stringPlayers().isEmpty()) {
        System.out.println("No players.");
        System.out.println();
    } else {
        System.out.println(Server.stringPlayers());
    }
```

When a client disconnects…

18.

```
public void giveBall(int received, int currentID) {
    // validation
    if (doesClientExist(received) && received != currentID){ // if client exists and isnt him/herself
        System.out.println("Player: " + currentID + " passes ball to Player: " + received + ".");
        Server.gameState.replace(received, true); // replaces clientID's value with true
        removeBall(currentID);
    }
    else if (received == currentID){ // if player == player
        System.out.println("Player: " + received + " is throwing the ball back to him/herself...");
    }
    else{
        System.out.println("Player: " + received + " does not exist, returning ball!");
        Server.gameState.replace(currentID,true);
    }
}
```

This method utilizes validation whilst using if statements. It prevents the Client from passing the ball to a player that does not exist...

19. Due to personal time restraints no extra features were implemented.

Client Threads

Only one thread is running in the Client program. This thread is ran in the main method and is responsible for connecting to a socket, receiving and sending data.

Server Threads

Two threads are generated when running Server.java. One is generated in Server.java just by running the program, and one thread is manually started as a ClientHandler objet (one thread for every connection between Client-Server). This assigns each Client a separate thread in the Server, this is the reason why an infinite amount of Client can join the game. The ClientHandler thread is responsible for all data inputs/outputs between the Client and the Server (thus the name).

Java to C# Conversion (attempt)

An attempt was made and during conversion I noted some key differences that I had to make.

```
class Server
{
    public static Dictionary<TcpClient, int> ConnectedClients = new Dictionary<TcpClient, int>();
    public static SortedDictionary<int, bool> GameState = new SortedDictionary<int, bool>();


    public static TcpListener server;
    private static int _clientId = 0;
```

I soon learned that HashMaps and TreeMaps are not a part of the standard C# library, but can be replaced with Dictionaries. Sockets are also replaced by TcpListener and TcpClient.

A challenge that I faced was initializing the input and output streams and then passing them to the ClientHandler class. This wasn't clear to me as I did not understand how streams worked in C#, and how you could not reference them across the class as by nature the streams were relevant to each unique client. I think this issue came with not exactly knowing C# member access declarations.

In Server.cs

```
            Console.WriteLine(StringPlayers());

            Stream stream = client.GetStream();
            // creating instance object
            ClientHandler clientObj = new ClientHandler(stream, _clientId, client);

            // starting new thread using class
            Thread thread = new Thread(new ThreadStart(clientObj.ClientThread));
            thread.Start();
        }
        catch (Exception e)
```

(local variable) Stream stream

In ClientHandler.cs

```
    1 reference | 0 changes | 0 authors, 0 changes
    public ClientHandler (Stream clientStream, int clientId, TcpClient client)
    {
        this.ClientStream = clientStream;
        this.ClientId = clientId;
        this.Client = client;


        dos = new StreamWriter(ClientStream);
        dis = new StreamReader(ClientStream);
        dos.AutoFlush = true;
    }
```

I also managed to translate the timer implemented in java.

```
1 reference | 0 changes | 0 authors, 0 changes
public void SetTimer(System.Timers.Timer timer)
{

    timer.Elapsed+=new ElapsedEventHandler(OnTimedEvent);
    timer.Interval = Constants.Delay;
    timer.Enabled = true;
}

1 reference | 0 changes | 0 authors, 0 changes
private void OnTimedEvent(object source, ElapsedEventArgs e)
{

    string received;
    int receivedInt;
    try
```

Project Review

With all honesty I found this assignment very difficult. But, I do see the direct benefit of it. I believe that I improved my programming skill as I found myself programming at this task for hours on end. I managed to fill every requirement for Java mentioned in the assignment document. It was interesting to learn how to program with sockets, it was a topic I was always never considered learning about as it was so necessary in technology nowadays, it essentially flew over my head if that makes sense.

I am upset that I could not convert the program to C# as I spent a lot of time on it. I had managed to implement some functionality but not to my personal standard. I do think I need more practice with C#.

I am particularly proud of making the server send data over to the client every second with the use of an internal TimerTask class. The client interface as a result works exactly how I imagined it to. The Client sees the state of the game every second – this is true for both the Client with the ball and without as the Client with the ball only needs to enter 0 to see a refreshed (refreshes every second) list whilst the other can just wait every second.

My time management for this project was very bad, and that is mainly because I over extended my efforts in other academic studies such as my dissertation. I will most definitely plan ahead and give myself more time for future projects such as these.