

## Completare laboratoare

## Lab 1. TCP

The screenshot displays an IDE window with a project named 'lab1'. The project structure is visible on the left, showing a 'src' directory with a 'main' package containing 'org' and 'lab1' sub-packages. The 'lab1' package contains 'TCPClient.java' and 'TCPServer.java' files. The 'TCPServer.java' file is open, showing a Java program that listens for client connections and echoes back the received data. The code is as follows:

```
while (true) {
    System.out.println("Awaiting client connection...");
    // wait for a client connection
    Socket clientSocket = serverSocket.accept();
    System.out.println("Client connected from " + clientSocket.getInetAddress());
    dataInputStream = new DataInputStream(clientSocket.getInputStream());
    dataOutputStream = new DataOutputStream(clientSocket.getOutputStream());

    // communicate with the client
    try {
        dataOutputStream.writeUTF("Welcome to the TCP Echo Server!");
        String input;
        while (true) {
            SimpleDateFormat d=new SimpleDateFormat("dd.MM.yyyy");
            SimpleDateFormat h=new SimpleDateFormat("HH:mm:ss");
            input = (h.format(new Date()))+" "+dataInputStream.readUTF();
            System.out.println(d.format(new Date())+" "+input);
            dataOutputStream.writeUTF("Received by server at "+input);
        }
    } catch (IOException e) {
        System.out.println("Client disconnected from server");
    } finally {
        // Close the client socket's streams
        dataInputStream.close();
        dataOutputStream.close();
        // Close the client socket
    }
}
```

The Run console at the bottom shows the output of the program:

```
Server created on port 3000
Awaiting client connection...
Client connected from /127.0.0.1
14.02.2024 01:33:35 Test
```

The screenshot displays the IntelliJ IDEA IDE interface. The top toolbar shows standard development icons. The left sidebar contains the Project view, showing the project structure for 'lab1' (D:\Desktop\lab1\lab1). The main editor window shows the source code for 'TCPClient.java'. The code defines a 'TCPClient' class with a 'main' method that establishes a socket connection to a server at 'localhost' on port '8080'. It sets up input and output streams and a 'BufferedReader' for reading from the server. The 'main' method includes a loop to read data from the server and print it to the console. The 'Run' tab at the bottom shows the execution output, indicating a successful connection and data exchange between the client and server.

```
1  if (args.length == 0) {
2      System.out.println("Usage: java TCPClient <server_address> <port>");
3      return;
4  }
5
6  Socket socket = null;
7  DataInputStream dataInputStream = null;
8  DataOutputStream dataOutputStream = null;
9  BufferedReader keyboardReader = null;
10 // Connect to the server...
11 try {
12     socket = new Socket(args[0], Integer.parseInt(args[1]));
13     // Obtain the streams...
14     dataInputStream = new DataInputStream(socket.getInputStream());
15     dataOutputStream = new DataOutputStream(socket.getOutputStream());
16     keyboardReader = new BufferedReader(new InputStreamReader(System.in));
17 }
18 catch (IOException e) {
19     System.out.println("Problem initializing client: " + e);
20     System.exit(1);
21 }
22
23 // Start the listening thread...
24 TCPEchoReader reader = new TCPEchoReader(dataInputStream);
25 reader.setDaemon(true);
26 reader.start();
27 String input;
```

Run TCPServer x TCPClient x

D:\java\src-17\bin>java.exe -javaagent:D:\Program Files\JetBrains\IntelliJ IDEA 2023.2.5\lib\idea\_rt.jar64757:D:\Program Files\JetBrains\IntelliJ IDEA 2023.2.5\bin -Dfile.encoding=UTF-8 -classpath D:\Desktop\lab1\lab1\... 14.02.2024

Welcome to the TCP Echo Server! received back at 01:38:25

test

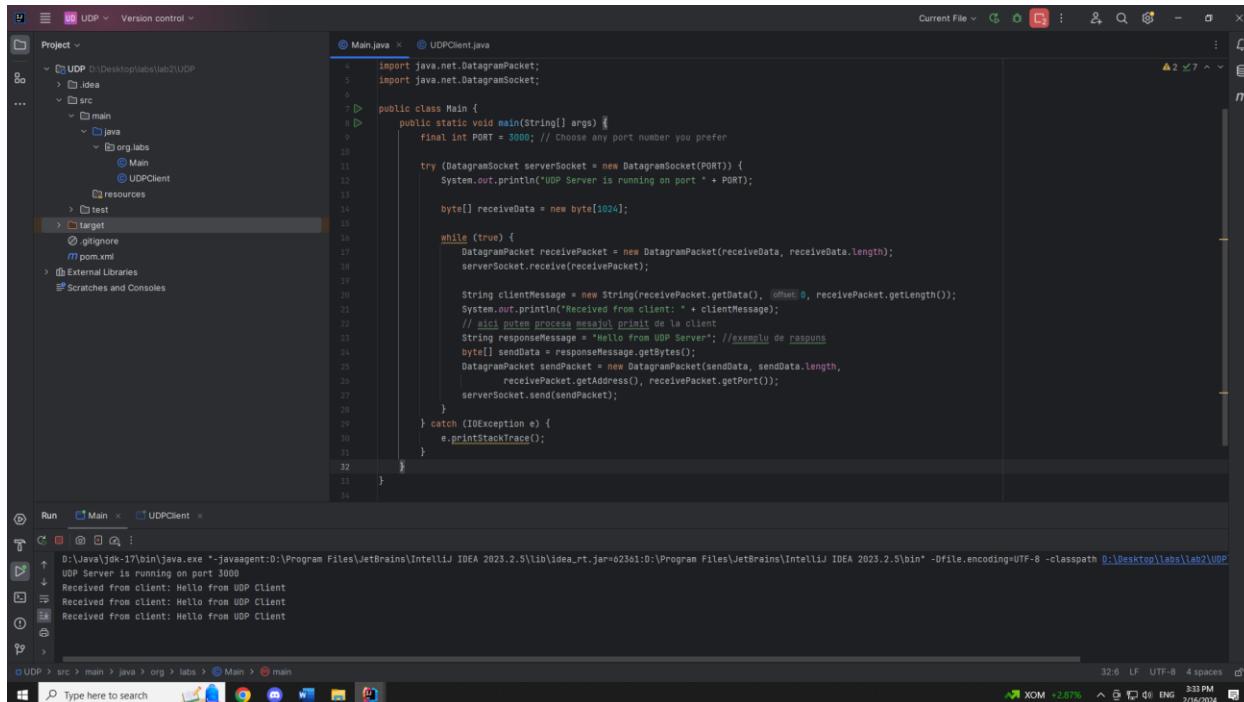
Received by server at 01:38:25 test received back at 01:39:27

Am modificat exemplul de echo server astfel incat serverul sa nu se inchida daca userul/userii se deconecteaza.

Am modificat si formatul mesajelor pentru a putea calcula timpul mediu de raspuns al serverului.

Serverul inregistreaza in consola ora la care s-a receptionat mesajul si apoi trimite sub forma unui ecou spre client mesajul precedat de ora la care a fost receptionat. Ulterior threadul de citire al ecolui accepta datele trimise de server si adauga la final ora la care a fost receptionat pe client mesajul.

Lab 1. UDP – aparent il mai facusem in trecut, dar nu l-am incarcat



```
import java.net.DatagramPacket;
import java.net.DatagramSocket;

public class Main {
    public static void main(String[] args) {
        final int PORT = 3000; // Choose any port number you prefer

        try (DatagramSocket serverSocket = new DatagramSocket(PORT)) {
            System.out.println("UDP Server is running on port " + PORT);

            byte[] receiveData = new byte[1024];

            while (true) {
                DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
                serverSocket.receive(receivePacket);

                String clientMessage = new String(receivePacket.getData(), 0, receivePacket.getLength());
                System.out.println("Received from client: " + clientMessage);
                // aici putem procesa mesajul primit de la client
                String responseMessage = "Hello from UDP Server"; // exemplu de raspuns
                byte[] sendData = responseMessage.getBytes();
                DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
                    receivePacket.getAddress(), receivePacket.getPort());
                serverSocket.send(sendPacket);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Run Main - UDPClient

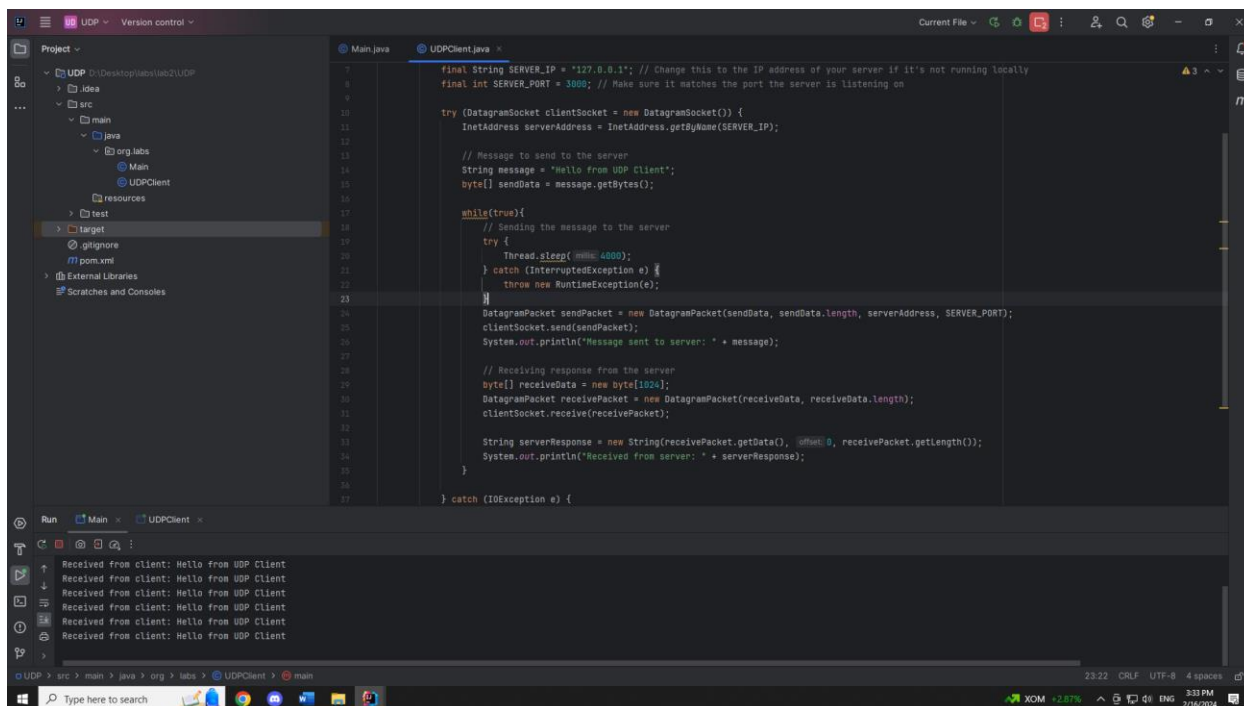
D:\Java\jdk-17\bin\java.exe -javaagent:D:\Program Files\JetBrains\IntelliJ IDEA 2023.2.5\lib\idea\_rt.jar=62361:0:\Program Files\JetBrains\IntelliJ IDEA 2023.2.5\bin -Dfile.encoding=UTF-8 -classpath D:\Desktop\lab1\lab2\UDP

UDP Server is running on port 3000

Received from client: Hello from UDP Client

Received from client: Hello from UDP Client

Received from client: Hello from UDP Client



```
final String SERVER_IP = "127.0.0.1"; // Change this to the IP address of your server if it's not running locally
final int SERVER_PORT = 3000; // Make sure it matches the port the server is listening on

try (DatagramSocket clientSocket = new DatagramSocket()) {
    InetAddress serverAddress = InetAddress.getByName(SERVER_IP);

    // Message to send to the server
    String message = "Hello from UDP Client";
    byte[] sendData = message.getBytes();

    while (true) {
        // Sending the message to the server
        try {
            Thread.sleep(4000);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }

        DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, serverAddress, SERVER_PORT);
        clientSocket.send(sendPacket);
        System.out.println("Message sent to server: " + message);

        // Receiving response from the server
        byte[] receiveData = new byte[1024];
        DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
        clientSocket.receive(receivePacket);

        String serverResponse = new String(receivePacket.getData(), 0, receivePacket.getLength());
        System.out.println("Received from server: " + serverResponse);
    }
} catch (IOException e) {
    e.printStackTrace();
}
```

Run Main - UDPClient

Received from client: Hello from UDP Client

Received from client: Hello from UDP Client

Received from client: Hello from UDP Client

Received from client: Hello from UDP Client

Received from client: Hello from UDP Client

Un protocol de comunicare pentru transmiterea unor mesaje sub forma unor siruri de octeti.

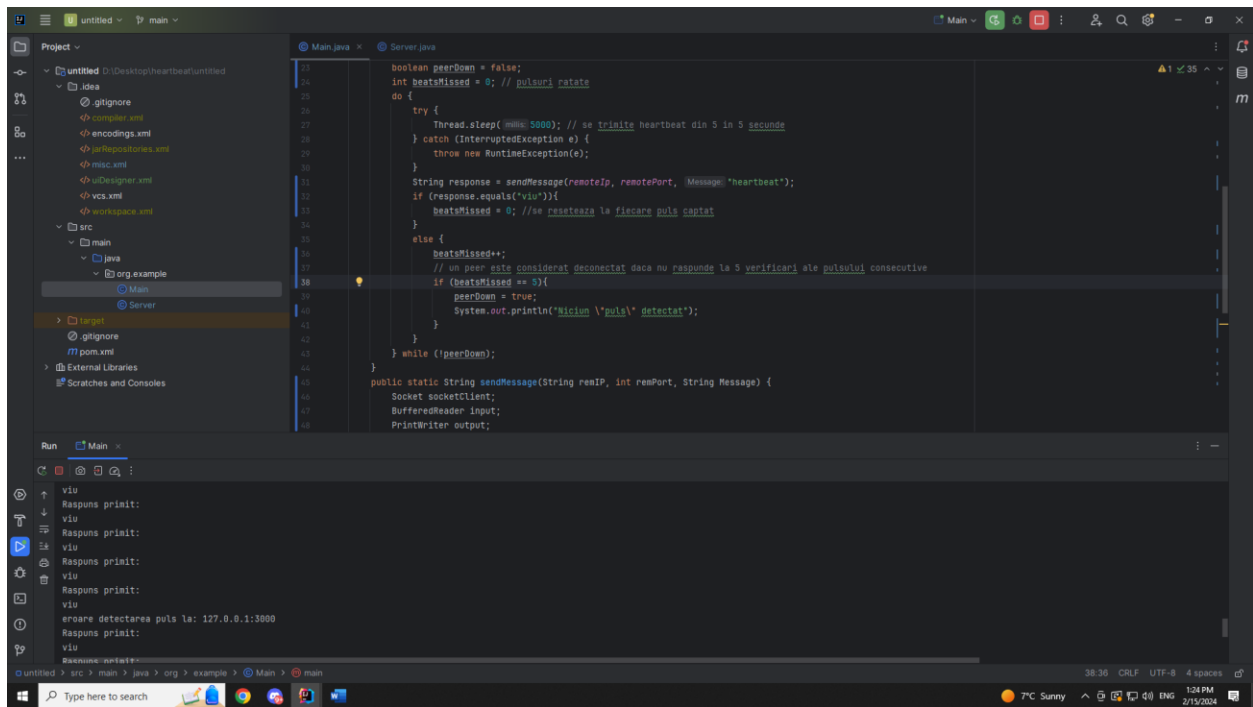
## Lab 2. HeartBeat – inspiratie de pe github de la un coleg

```
14 if (beatsMissed == 5){
15     peerDown = true;
16     System.out.println("Niciun \"puls\" detectat");
17 }
18 } while (!peerDown);
19
20 public static String sendMessage(String remIP, int remPort, String Message) {
21     Socket socketClient;
22     BufferedReader input;
23     PrintWriter output;
24     try {
25         // socket cu peer-ul invecinat
26         socketClient = new Socket(remIP, remPort);
27         // reader si writer pentru manipulara fluxului de date de intrare sau iesire prin socket
28         input = new BufferedReader(new InputStreamReader(socketClient.getInputStream()));
29         output = new PrintWriter(socketClient.getOutputStream(), autoFlush: true);
30
31         output.println(Message);
32         String raspuns = input.readLine();
33
34         System.out.println("Raspuns primit:");
35         System.out.println(raspuns);
36         return raspuns;
37     } catch (IOException e) {
38         System.out.println("eroare detectarea puls la: " + remIP + ":" + remPort);
39     }
40 }
```

```
1 //program "serverului" vine "at pulsul" peer
2 new Server().start();
3 boolean peerDown = false;
4 int beatsMissed = 0; // pulsuri ratate
5 do {
6     try {
7         Thread.sleep(5000); // se trimite heartbeat din 5 in 5 secunde
8     } catch (InterruptedException e) {
9         throw new RuntimeException(e);
10    }
11    String response = sendMessage(remoteIP, remotePort, "heartbeat");
12    if (response.equals("vio")){
13        beatsMissed = 0; //se reseteaza la fiecare puls castat
14    }
15    else {
16        beatsMissed++;
17        // un peer este considerat deconectat daca nu raspunde la 5 verificari ale pulsului consecutive
18        if (beatsMissed == 5){
19            peerDown = true;
20            System.out.println("Niciun \"puls\" detectat");
21        }
22    }
23 } while (!peerDown);
24
25 public static String sendMessage(String remIP, int remPort, String Message) {
26     Socket socketClient;
27     BufferedReader input;
28     PrintWriter output;
29     try {
30         // socket cu peer-ul invecinat
31         socketClient = new Socket(remIP, remPort);
32         // reader si writer pentru manipulara fluxului de date de intrare sau iesire prin socket
33         input = new BufferedReader(new InputStreamReader(socketClient.getInputStream()));
34         output = new PrintWriter(socketClient.getOutputStream(), autoFlush: true);
35
36         output.println(Message);
37         String raspuns = input.readLine();
38
39         System.out.println("Raspuns primit:");
40         System.out.println(raspuns);
41         return raspuns;
42     } catch (IOException e) {
43         System.out.println("eroare detectarea puls la: " + remIP + ":" + remPort);
44     }
45 }
```

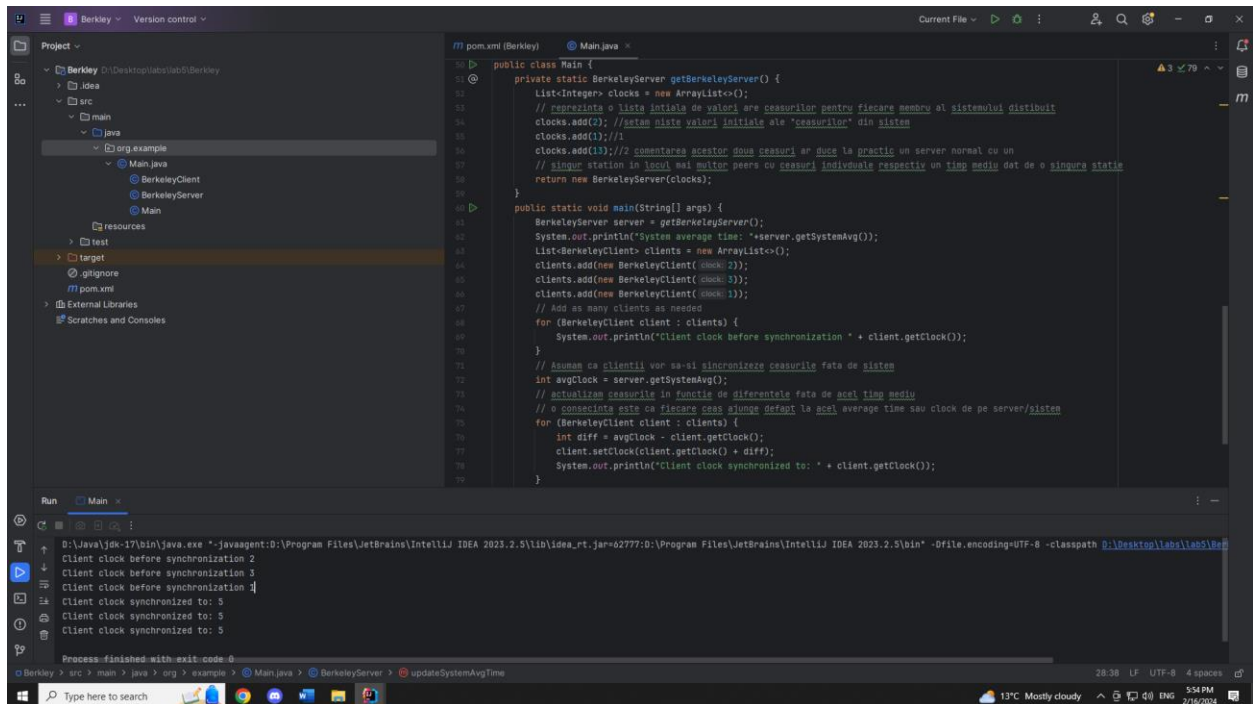
Daca ambi clienti sunt porniti de odata avem raspunsuri de viu pe fiecare parte.

Daca un peer se deconecteaza celalalt v-a mai verifica pulsul de 5 ori si ulterior v-a mentiona lipsa pulsului(daca nu s-a reconectat).



O implementare mai interesanta ar fi crearea unui hash map in care sa tinem valorile de porturi si ip-uri sub forma unor perechi cheie-valoare. Unde ip-uri ar fi cheile, si porturile ar fi valorile. Instantierea clientilor ar fi pe thread-uri diferite realizata intr-o bucla pentru parcurgerea fiecarei perechi din hash map.

## Lab 4. Berkeley

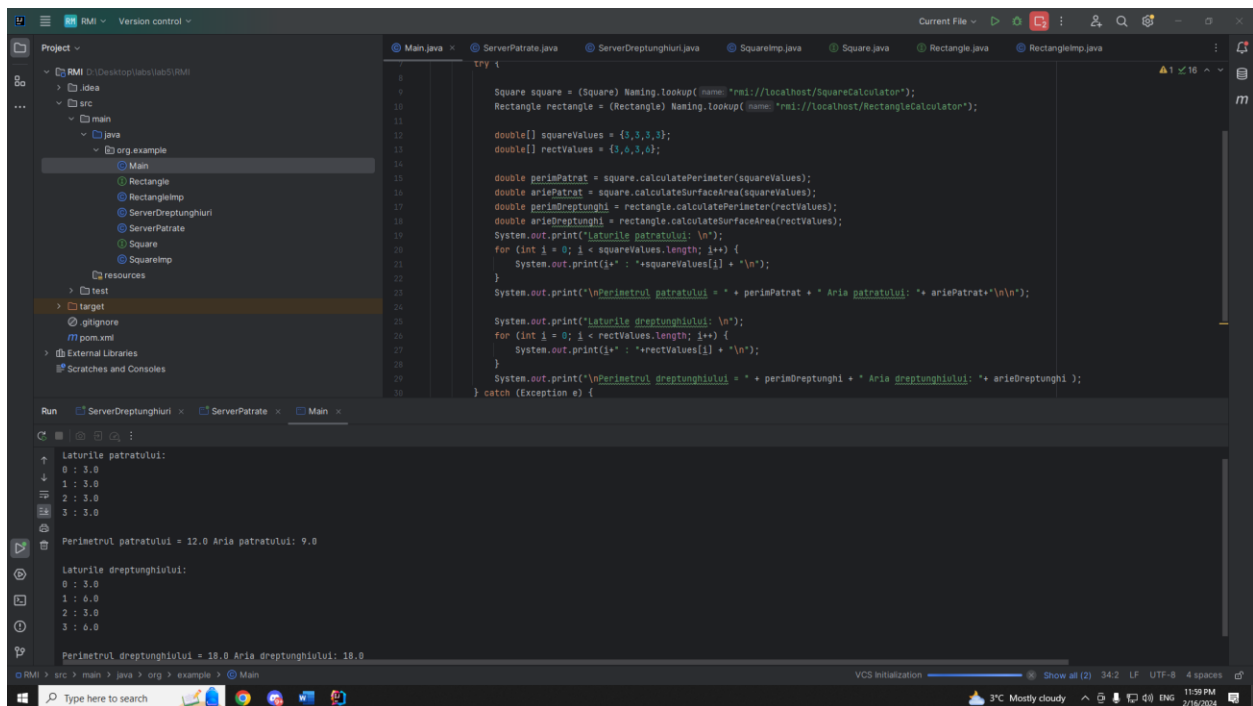


Din ce am inteles eu, acest algoritm sincronizeaza atat ora din sistem cat si orele de pe celelalte noduri. Initial se calculeaza un averageTime la nivel de sistem si ulterior se calibreaza fiecare nod adaugand la el diferenta dintre ora medie si ora nodului, rezultat care il aduce la acea ora medie. Ducand astfel fiecare nod la sincronizare.

Fiecare ceas fiind preferabil fie incetinit, fie oprit temporar pentru a ajunge la orele potrivite in loc de a fi setat la o anumita ora (cum am facut eu).

## Lab 5. RMI

### Remote Method Invocation



The screenshot shows an IDE with a project named 'RMI'. The project structure includes a 'src' directory with 'org.example' containing 'Main', 'Rectangle', 'RectangleImp', 'ServerDreptunghiuri', 'ServerPatrat', 'Square', and 'SquareImp'. The 'Main' class is selected, and its code is visible in the editor. The code uses RMI to connect to remote services 'SquareCalculator' and 'RectangleCalculator' on 'localhost'. It calculates the perimeter and area for a square and a rectangle, then prints the results. The console output shows the following:

```
Laturile patratului:
0 : 3.0
1 : 3.0
2 : 3.0
3 : 3.0

Perimetrul patratului = 12.0 Aria patratului: 9.0

Laturile dreptunghiului:
0 : 3.0
1 : 6.0
2 : 3.0
3 : 6.0

Perimetrul dreptunghiului = 18.0 Aria dreptunghiului: 18.0
```

Remote Method Invocation (RMI) este un api java care permite unui program să apeleze metodele unui obiect care se află pe un alt calculator într-o rețea.

<https://github.com/balintcristian/labs.git>