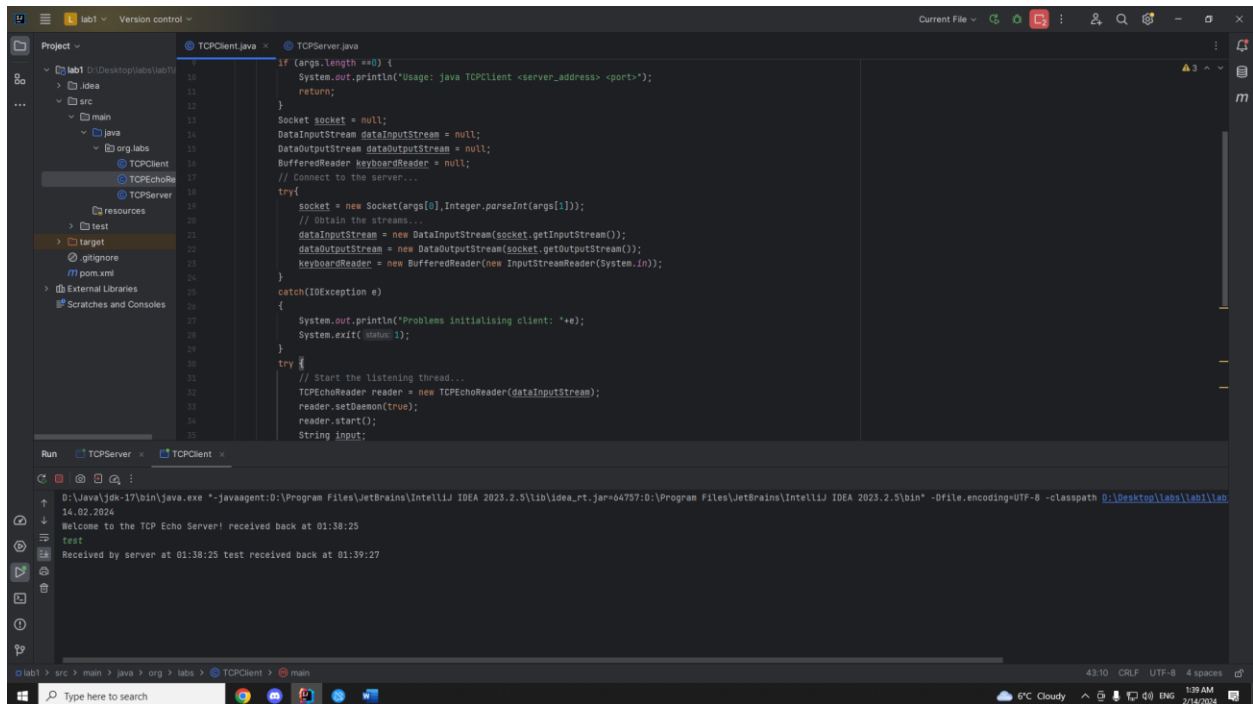


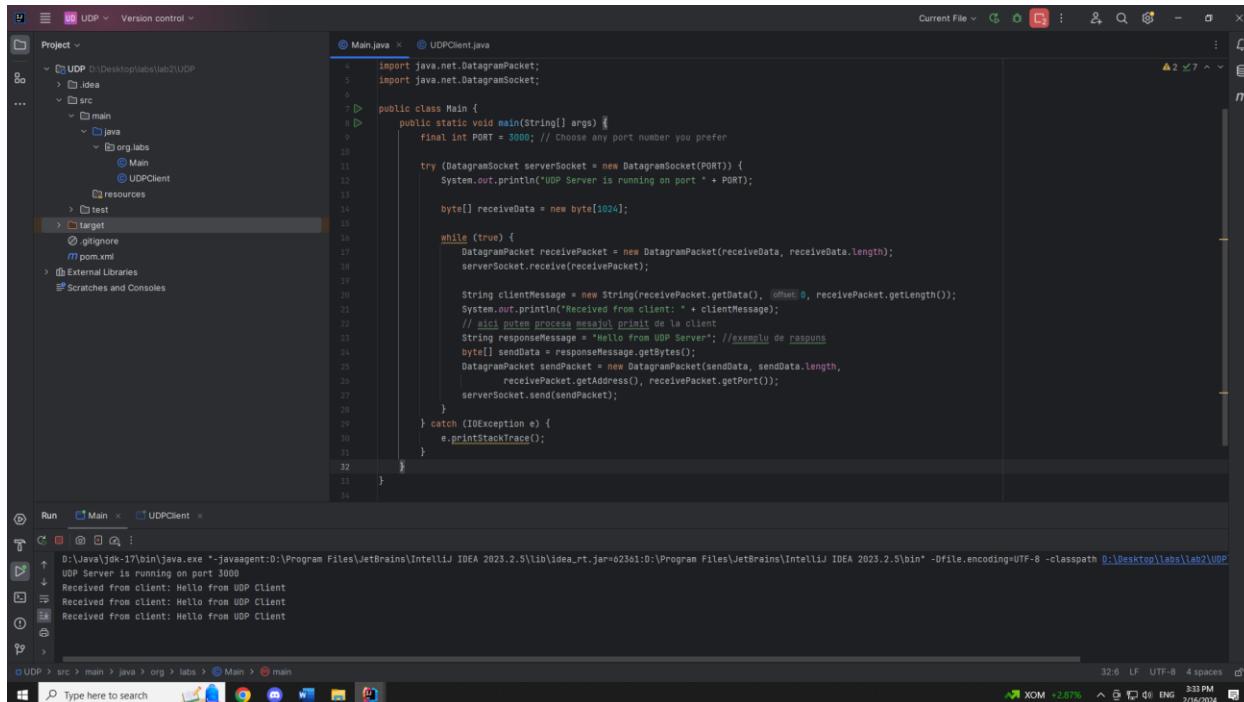
Lab 1. TCP



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.

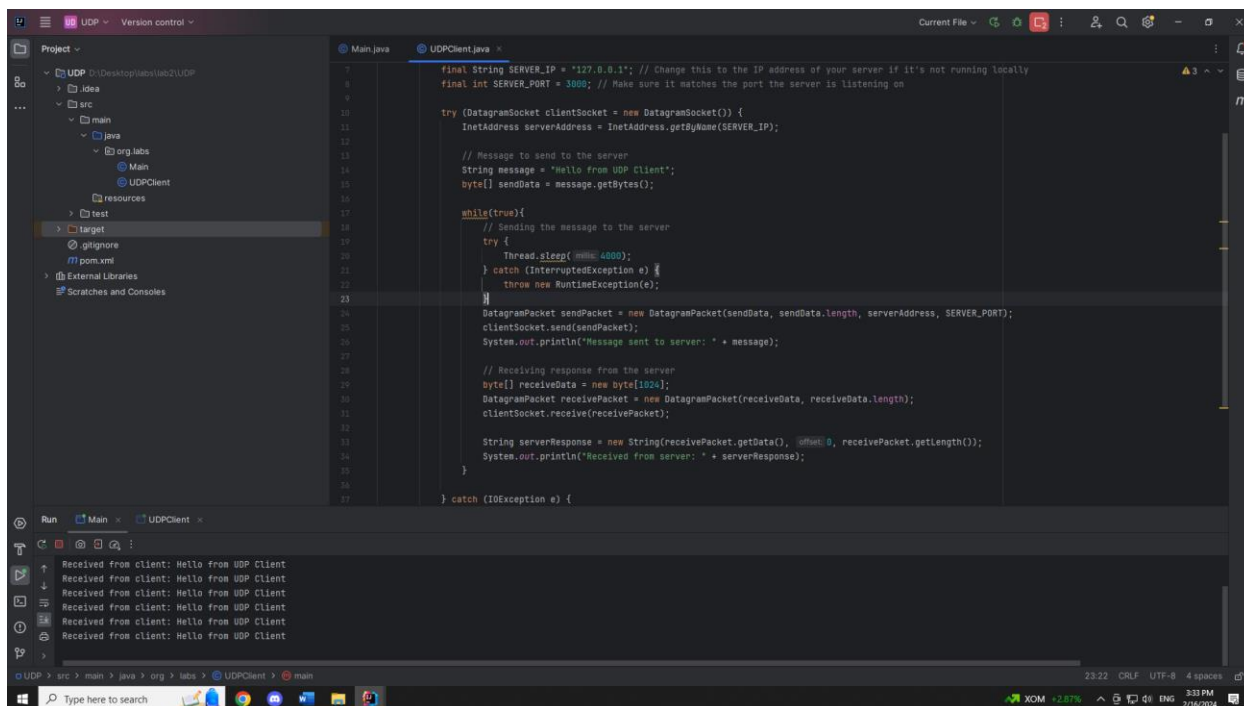
Lab 1. UDP – aparent il mai facusem in trecut, dar nu l-am incarcat



```
1 import java.net.DatagramPacket;
2 import java.net.DatagramSocket;
3
4 public class Main {
5     public static void main(String[] args) {
6         final int PORT = 3000; // Choose any port number you prefer
7
8         try (DatagramSocket serverSocket = new DatagramSocket(PORT)) {
9             System.out.println("UDP Server is running on port " + PORT);
10
11             byte[] receiveData = new byte[1024];
12
13             while (true) {
14                 DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
15                 serverSocket.receive(receivePacket);
16
17                 String clientMessage = new String(receivePacket.getData(), 0, receivePacket.getLength());
18                 System.out.println("Received from client: " + clientMessage);
19                 // aici putem procesa mesajul primit de la client
20                 String responseMessage = "Hello from UDP Server"; // exemplu de raspuns
21                 byte[] sendData = responseMessage.getBytes();
22                 DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
23                     receivePacket.getAddress(), receivePacket.getPort());
24                 serverSocket.send(sendPacket);
25             }
26         } catch (IOException e) {
27             e.printStackTrace();
28         }
29     }
30 }
```

Run console output:

```
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
```



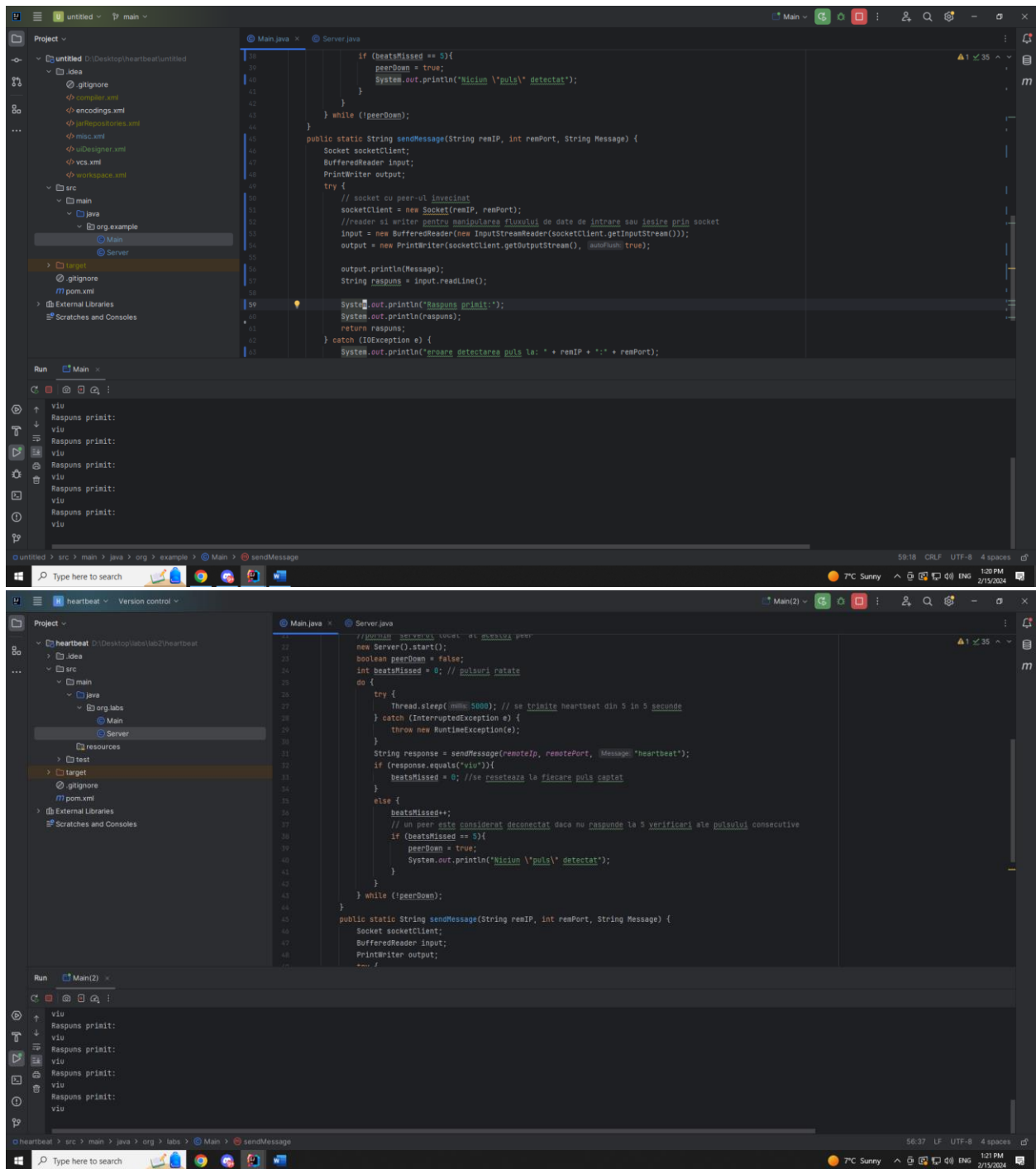
```
1 final String SERVER_IP = "127.0.0.1"; // Change this to the IP address of your server if it's not running locally
2 final int SERVER_PORT = 3000; // Make sure it matches the port the server is listening on
3
4 try (DatagramSocket clientSocket = new DatagramSocket()) {
5     InetAddress serverAddress = InetAddress.getByName(SERVER_IP);
6
7     // Message to send to the server
8     String message = "Hello from UDP Client";
9     byte[] sendData = message.getBytes();
10
11     while (true) {
12         // Sending the message to the server
13         try {
14             Thread.sleep(4000);
15         } catch (InterruptedException e) {
16             throw new RuntimeException(e);
17         }
18         DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, serverAddress, SERVER_PORT);
19         clientSocket.send(sendPacket);
20         System.out.println("Message sent to server: " + message);
21
22         // Receiving response from the server
23         byte[] receiveData = new byte[1024];
24         DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
25         clientSocket.receive(receivePacket);
26
27         String serverResponse = new String(receivePacket.getData(), 0, receivePacket.getLength());
28         System.out.println("Received from server: " + serverResponse);
29     }
30 } catch (IOException e) {
31     e.printStackTrace();
32 }
```

Run console output:

```
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
Received from client: Hello from UDP Client
```

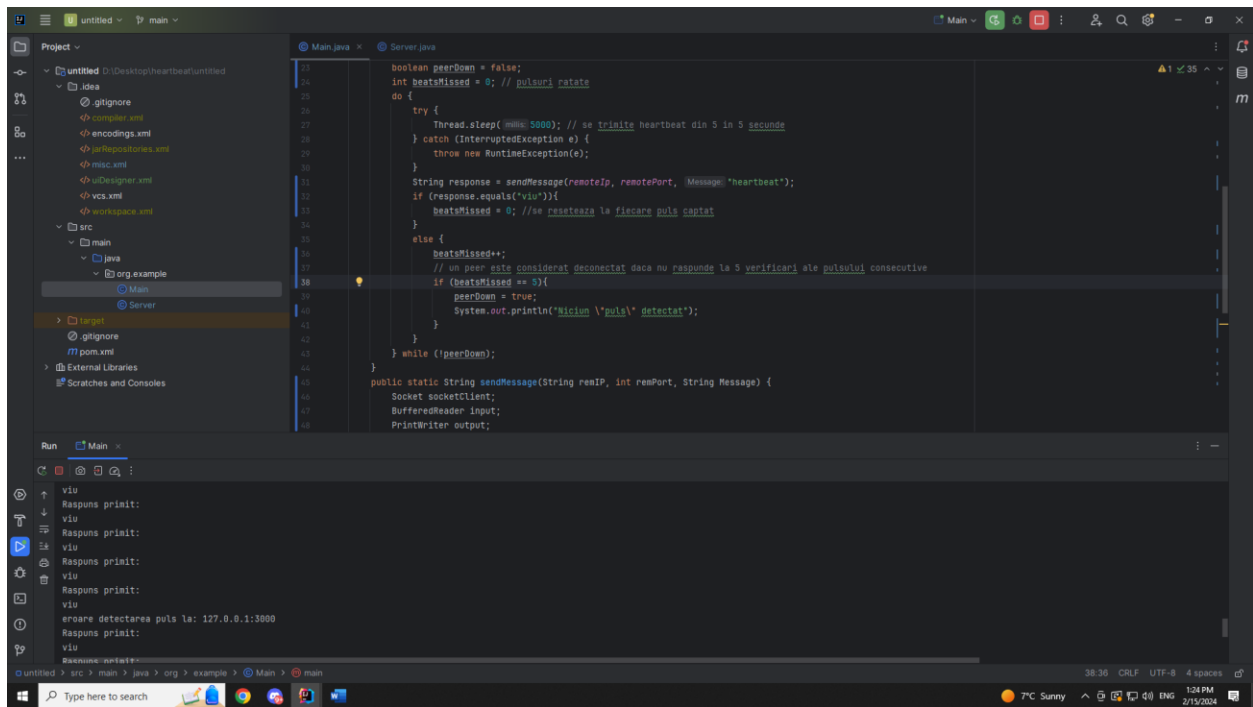
Un protocol de comunicare pentru transmiterea unor mesaje sub forma unor siruri de octeti.

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



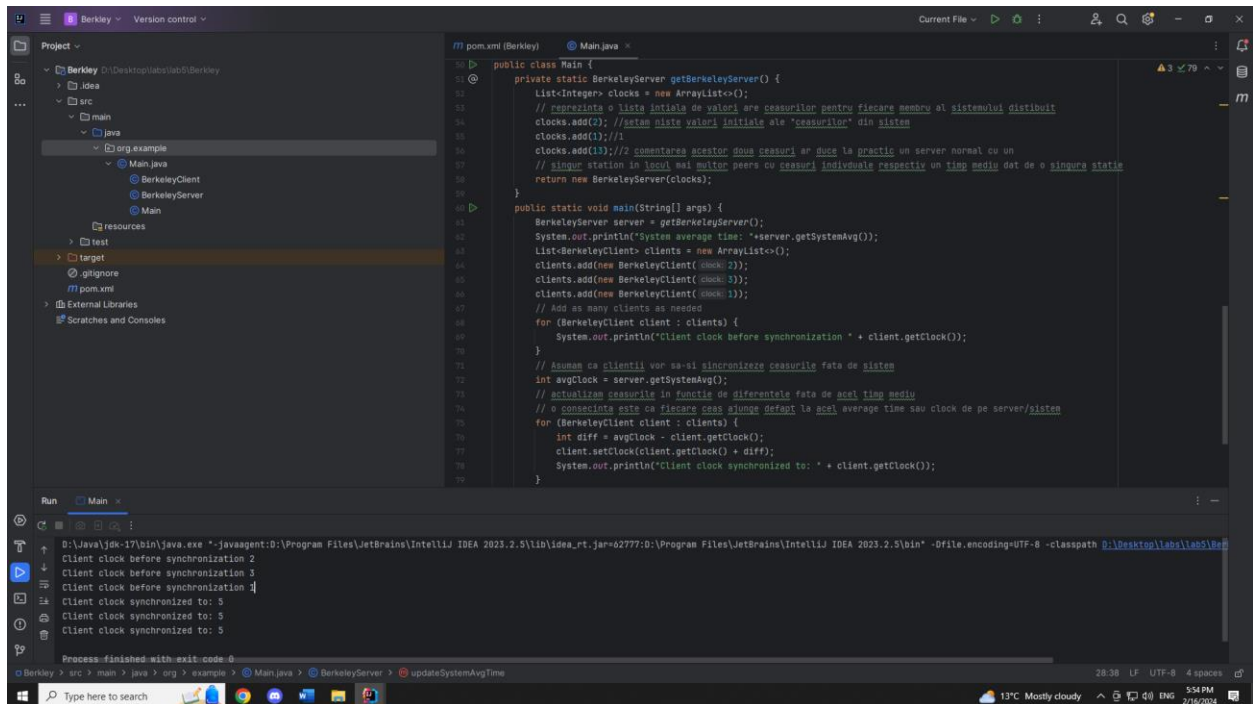
Daca ambi clienti sunt porniti de odata avem raspunsuri de viu pe fiecare parte.

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

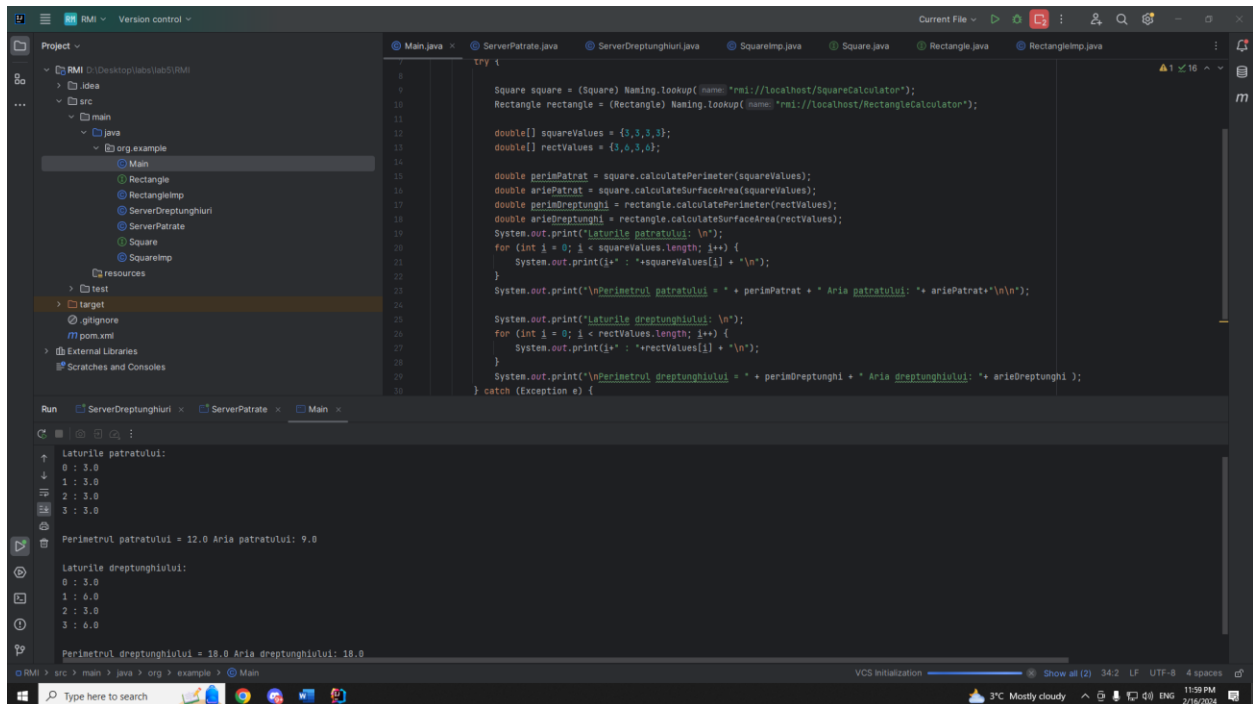


Din ce am inteles eu, acest algoritm sincronizeaza atat ora din sistem cat si orele de pe celelalte noduri. Initial se calculeaza un averageTime la nivel de sistem si ulterior se calibreaza fiecare nod adaugand la el diferenta dintre ora medie si ora nodului, rezultat care il aduce la acea ora medie.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

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

Perimetrul patratului = 12.0 Aria patratului: 9.0

Latunile 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.