DJILLALI LIABES UNIVERSITY OF SIDI BEL ABBES
FACULTY OF EXACT SCIENCES
DEPARTMENT OF COMPUTER SCIENCES

Module : Réseaux et Systèmes Répartis
1ST YEAR OF MASTER'S DEGEREE IN
NETWORKS,INFORMATION SYSTEMS & SECURITY(RSSI)
2021/2022

Application client-serveur avec les sockets TCP java pour la recherche de doublons dans un tableau

Students:
HADJAZI M.Hisham
AMOUR Wassim Malik
Group: 01/RSSI

Instructors:
Dr. MEHADJI Djamil

A paper submitted in fulfilment of the requirements for the
TP-01

March 2, 2022

Contents

Chapter 1

Application client-serveur avec les sockets TCP java

1.1 Introduction

For data to flow across the internet, a large number of protocols have been defined to maintain uniformity and reliability. This allows data to flow across billions of possible routes from source to the destination. Transmission Control Protocol (TCP) along with Internet Protocol (IP) define the crux of transmission of data across the internet through URL connections. Many a times, we require to send data locally within applications, or closely connected peers. Java Sockets are used precisely for this use case among others.[2]

TCP is mainly used in cases where 100% reliability of connection is of utmost importance. In order to achieve the same, TCP implements a large number of protocols, like congestion control, flow control etc.. However, for a TCP connection to be established, it first requires the source and destination ports, to implement sockets at both ends and bind them throughout the process of communication. While accessing websites, all of this happens through predefined logic, however when implementing custom logics relevant to one's use case one must use specific libraries to implement such sockets and communication logic.[2]

Client-Server Architecture is the most prominent application structure on the Internet. In this architecture, clients (eg: personal computers, IoT devices, etc.) first request resources from a server. Then the server sends back appropriate responses for the clients' requests. For this to happen, there should be some mechanism implemented in both the clients and the servers which supports this network transaction. That mechanism is called socket communication.[1]

Almost every application which relies on the network operations, such as fetching data from remote servers and uploading files to the server, extensively utilize sockets under the hood. Several examples of such applications are Browsers, chat applications, and Peer to Peer networking applications.[1]

1.1.1 What is a socket, exactly?

A socket is a "software" thing. In other words, a socket doesn't exist physically. An application software defines a socket so that it utilizes ports in the underlying computer for its implementation. This enables programmers to comfortably deal with the low-level details of the network communication such as ports, routing, etc inside their application code.[1]

To get you through this I'm gonna develop simple client and server programs. And I will make them talk to each other. For all these client-server connections I need sockets. So that's a quick overview to give you a brief idea. Let's get started.[3]

First let's consider clients. There are 3 things we have to do with client programs.[3]

1. How to establish the initial connection between the client and the server.
2. How to send messages to the server.
3. How to receive messages from the server.

To make the connection with a server we need a Socket connection. A Socket means an object that comes with java.net.Socket class. It is used to represent a network connection between 2 machines. To make this Socket object we need the IP address of the machine and TCP port number. An IP address identifies a host/computer on a computer network that uses the Internet Protocol for communication and a port is a communication endpoint. There is a specific port number for each process in an operating system. As port numbers from 0 to 1023 are reserved for well-known services such as HTTP, FTP, POP3, Telnet, etc we can use any other port number for our programs between 1024 and 65535.[3]

1.1.2 How do sockets work?

The TCP socket communication between a client and the server consists of several phases.

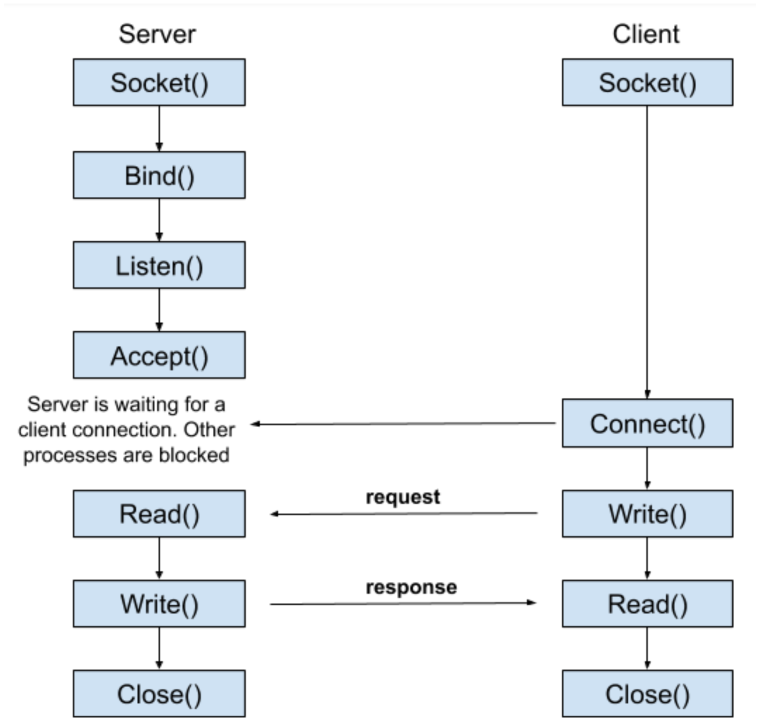


FIGURE 1.1: TCP Socket Communication Flow Diagram.

1. **Socket()** An endpoint for communication is created in the server.
2. **Bind()** Assigning a unique number to the socket and reserving a unique Combination of IP address & port for the created socket.
3. **Listen()** After creating a socket, the server is waiting for a client to connect.
4. **Accept()** The server receives a connection request from a client socket.
5. **Connect()** The client and the server are connected with each other.
6. **Send(), Recieve()** Exchanging data between the client and the server.
7. **Close()** After data exchange, the server and the client hangs up the connection.

Each phase of the socket communication listed above, have a lot of complex things going on under the hood. However, this knowledge is well enough for the sake of understanding and demonstrating how TCP socket communication works.[1]

1.2 Question 1

Le client envoie un tableau d'entiers au serveur. Le serveur recherche si le tableau reçu contient des doublons (des éléments avec la même valeur) et envoie la réponse au client. Par exemple :

1.2.1 Si le client envoie 12, 30, 5, 5,100 le serveur répond : »votre tableau contient un doublon ».

Client Sending Table

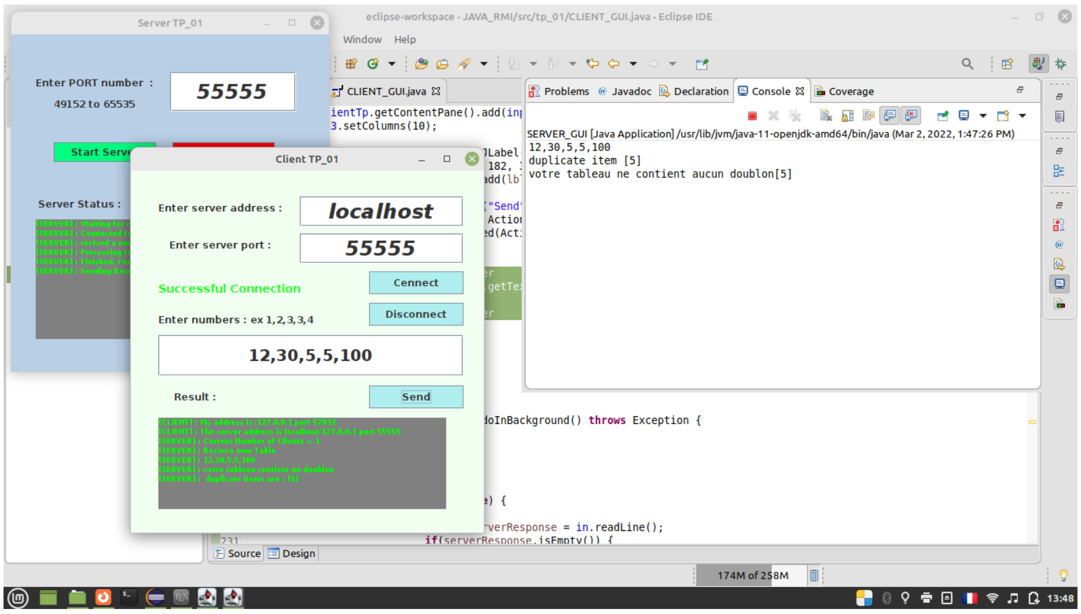
```
//                getting input from user
                  String array = input3.getText();

//                sending input to server
                  out.println(array);
```

Finding Repeated elements algorithm.

```
                  for (int i = 0; i < array.length-1; i++) {

                      if (array[i] == array[i+1]) {
                          System.out.println("duplicate item ["+array
                              [i+1]+"]");
                          record.add(String.valueOf(array[i]));

                      }

                  }
```

Server sending Positive Response.

```
if (record.isEmpty()) {
                      out.println("votre tableau ne contient aucun
                          doublon\n");
                      System.out.println("votre tableau ne contient
                          aucun doublon");
                      SERVER_GUI.txtout.append("[SERVER] : Finished
                          with NO Repeated elements \n");
                  }
```



```
[SERVER] : Waiting for c
[SERVER] : Connected t
[SERVER] : recived a ne
[SERVER] : Processing t
[SERVER] : Finished. Fo
[SERVER] : Sending Res
```

```
[CLIENT] : My address is /127.0.0.1 port 57032
[CLIENT] : The server address is localhost/127.0.0.1 port 55555
[SERVER] : Current Number of Clients = 1
[SERVER] : Recieve new Table
[SERVER] : 12,30,5,5,100
[SERVER] : votre tableau contient un doublon
[SERVER] :  duplicate items are : [5]
```

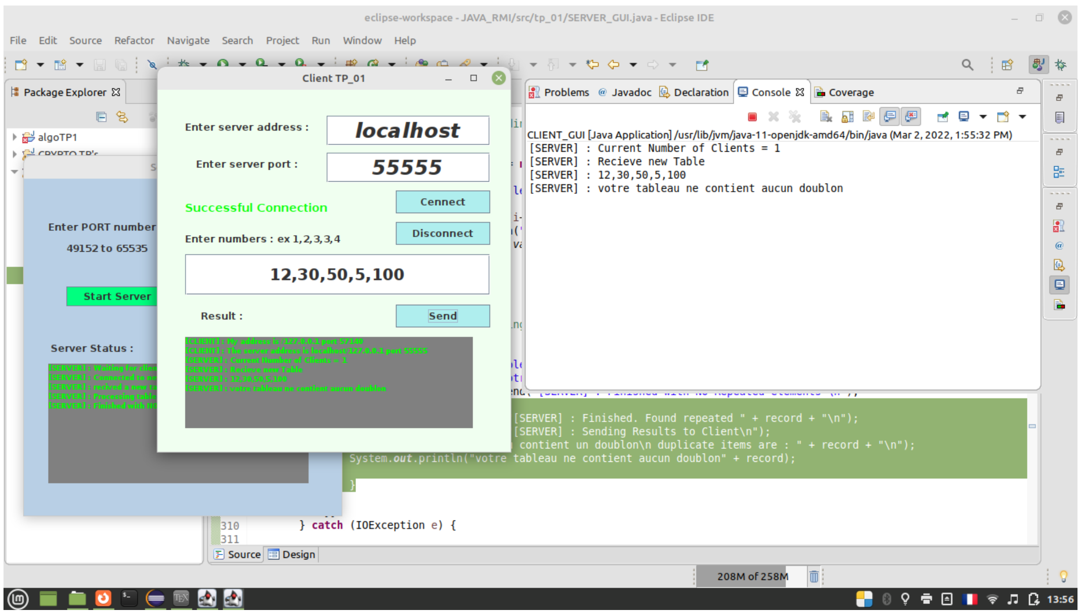
```
SERVER_GUI [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (Mar 2, 2022, 1:47:26 PM)
12,30,5,5,100
duplicate item [5]
votre tableau ne contient aucun doublon[5]
```

FIGURE 1.2: found duplicates.

1.2.2 Si le client envoie 12, 30, 50, 5,100 le serveur répond : »votre tableau ne contient aucun doublon ».

Server sending Negative Response.

```
else {
                SERVER_GUI.txtout.append("[SERVER] : Finished.
                    Found repeated " + record + "\n");
                SERVER_GUI.txtout.append("[SERVER] : Sending
                    Results to Client\n");
                out.println("votre tableau contient un doublon\n
                    duplicate items are : " + record + "\n");
                System.out.println("votre tableau ne contient aucun
                     doublon" + record);

                }
```


eclipse-workspace - JAVA_RMI/src/tp_01/SERVER_GUI.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Client TP_01
Problems Javadoc Declaration Console Coverage
Cennect
Disconnect
Send
Start Server
Source Design
208M of 258M
13:56

Enter server address : localhost

Enter server port : 55555

Successful Connection

Enter numbers : ex 1,2,3,3,4

12,30,50,5,100

Result :

```
[CLIENT] : My address is /127.0.0.1 port 57140
[CLIENT] : The server address is localhost/127.0.0.1 port 55555
[SERVER] : Current Number of Clients = 1
[SERVER] : Recieve new Table
[SERVER] : 12,30,50,5,100
[SERVER] : votre tableau ne contient aucun doublon
```

Enter PORT number

49152 to 65535

Server Status :

```
[SERVER] : Waiting for clien
[SERVER] : Connected to ne
[SERVER] : recived a new ta
[SERVER] : Processing table
[SERVER] : Finished with NO
```

```
CLIENT_GUI [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (Mar 2, 2022, 1:55:32 PM)
[SERVER] : Current Number of Clients = 1
[SERVER] : Recieve new Table
[SERVER] : 12,30,50,5,100
[SERVER] : votre tableau ne contient aucun doublon
```

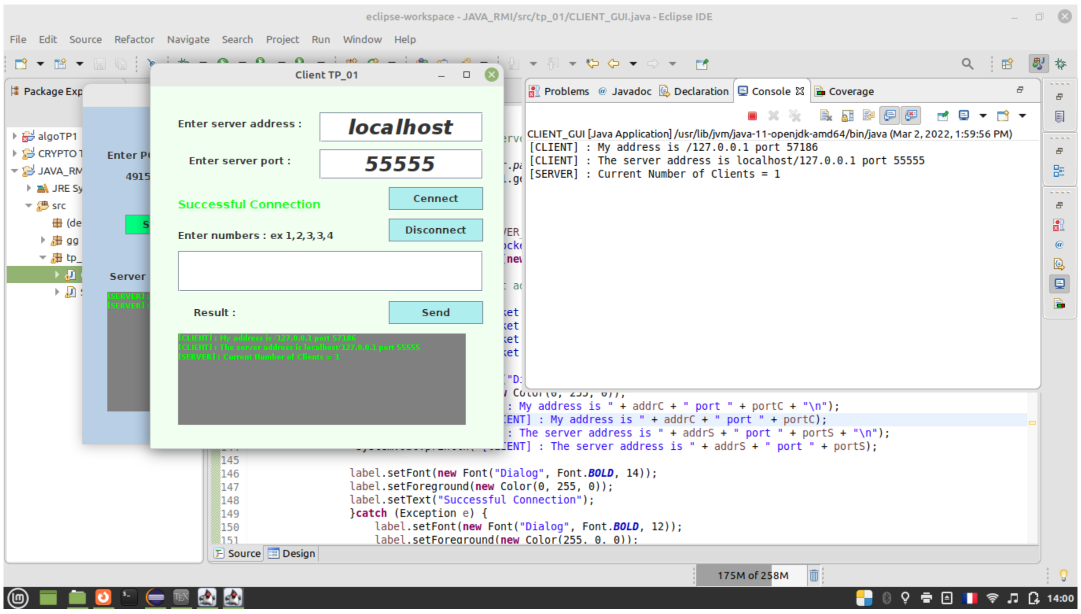
```
[SERVER] : Finished. Found repeated " + record + "\n");
[SERVER] : Sending Results to Client\n");
contient un doublon\n duplicate items are : " + record + "\n");
System.out.println("votre tableau ne contient aucun doublon" + record);
}
310            } catch (IOException e) {
311
```

FIGURE 1.3: no duplicates.

1.2.3 Le client doit afficher :

son adresse (port et adresse IP) et l'adresse du serveur (port et adresse IP)

```
//                 getting Sever and client address and port

                   InetAddress addrS = socket.getInetAddress();
                   InetAddress addrC = socket.getLocalAddress();
                   int         portS = socket.getPort();
                   int         portC = socket.getLocalPort();

                   txtout.setFont(new Font("Dialog", Font.BOLD, 9));
                   txtout.setForeground(new Color(0, 255, 0));
                   txtout.append("[CLIENT] : My address is " + addrC
                       + " port " + portC + "\n");
                   System.out.println("[CLIENT] : My address is " +
                       addrC + " port " + portC);
                   txtout.append("[CLIENT] : The server address is "
                       + addrS + " port " + portS + "\n");
                   System.out.println("[CLIENT] : The server address
                       is " + addrS + " port " + portS);
```

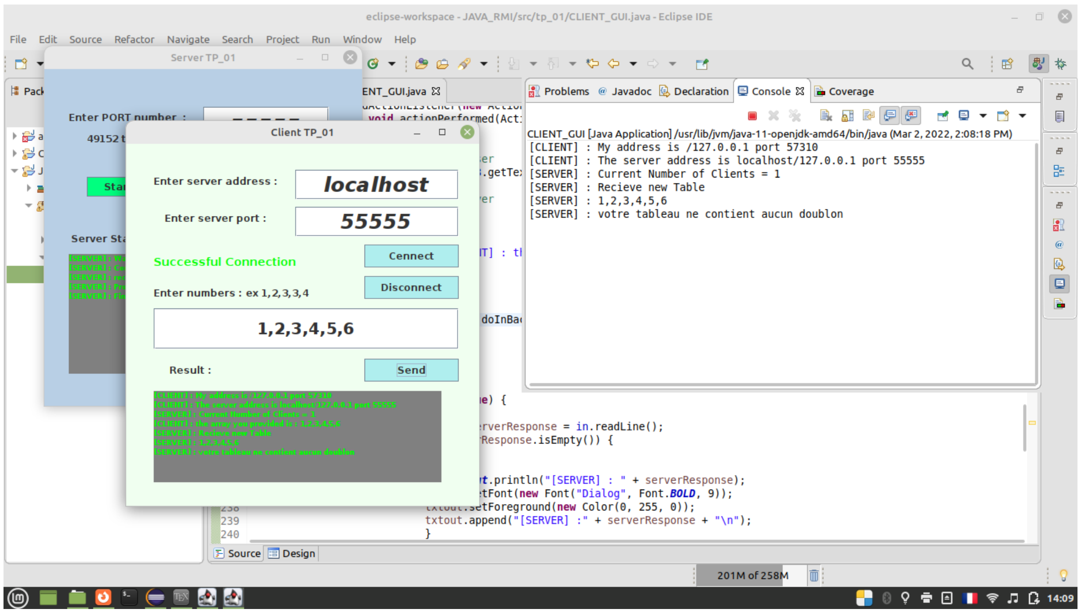


```
CLIENT_GUI [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (Mar 2, 2022, 1:59:56 PM)
[CLIENT] : My address is /127.0.0.1 port 57186
[CLIENT] : The server address is localhost/127.0.0.1 port 55555
[SERVER] : Current Number of Clients = 1
```

FIGURE 1.4: Showing client/server IP addresses and ports.

le tableau envoyé

```
//                  getting input from user
                    String array = input3.getText();

//                  sending input to server
                    out.println(array);

//                  printing table
                    txtout.append("[CLIENT] : the array you provided is
                        : " + array + "\n");
```



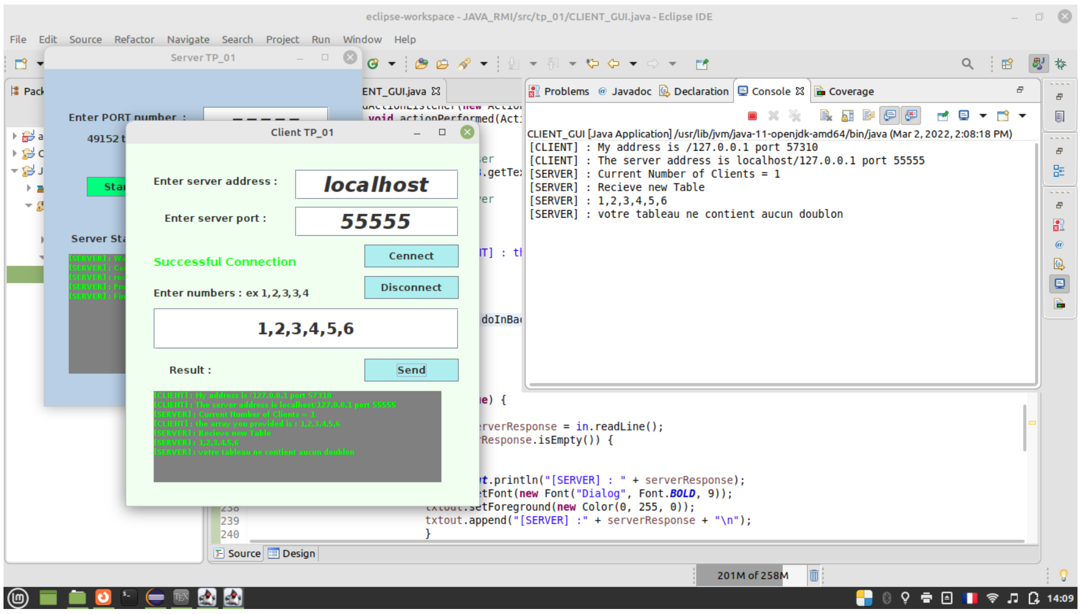
```
[CLIENT] : My address is /127.0.0.1 port 57310
[CLIENT] : The server address is localhost/127.0.0.1 port 55555
[SERVER] : Current Number of Clients = 1
[CLIENT] : the array you provided is : 1,2,3,4,5,6
[SERVER] : Recieve new Table
[SERVER] : 1,2,3,4,5,6
[SERVER] : votre tableau ne contient aucun doublon
```

```
CLIENT_GUI [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (Mar 2, 2022, 2:08:18 PM)
[CLIENT] : My address is /127.0.0.1 port 57310
[CLIENT] : The server address is localhost/127.0.0.1 port 55555
[SERVER] : Current Number of Clients = 1
[SERVER] : Recieve new Table
[SERVER] : 1,2,3,4,5,6
[SERVER] : votre tableau ne contient aucun doublon
```

FIGURE 1.5: Showing table.

la réponse du serveur

```
//                          Receiving solution from server

                            try {



                                while(true) {

                                String serverResponse = in.readLine();
                                if(serverResponse.isEmpty()) {
                                    // empty response
                                }else {
                                System.out.println("[SERVER] : " +
                                    serverResponse);
                                txtout.setFont(new Font("Dialog", Font.
                                    BOLD, 9));
                                txtout.setForeground(new Color(0, 255,
                                    0));
                                txtout.append("[SERVER] :" +
                                    serverResponse + "\n");
                                }

                                }



                                }catch(Exception e) {
                                    e.printStackTrace();
                                }
```


Enter server address : localhost

Enter server port : 55555

Successful Connection

Enter numbers : ex 1,2,3,3,4

1,2,3,4,5,6

Result :

```
[CLIENT] : My address is /127.0.0.1 port 57310
[CLIENT] : The server address is localhost/127.0.0.1 port 55555
[SERVER] : Current Number of Clients = 1
[CLIENT] : the array you provided is : 1,2,3,4,5,6
[SERVER] : Recieve new Table
[SERVER] : 1,2,3,4,5,6
[SERVER] : votre tableau ne contient aucun doublon
```

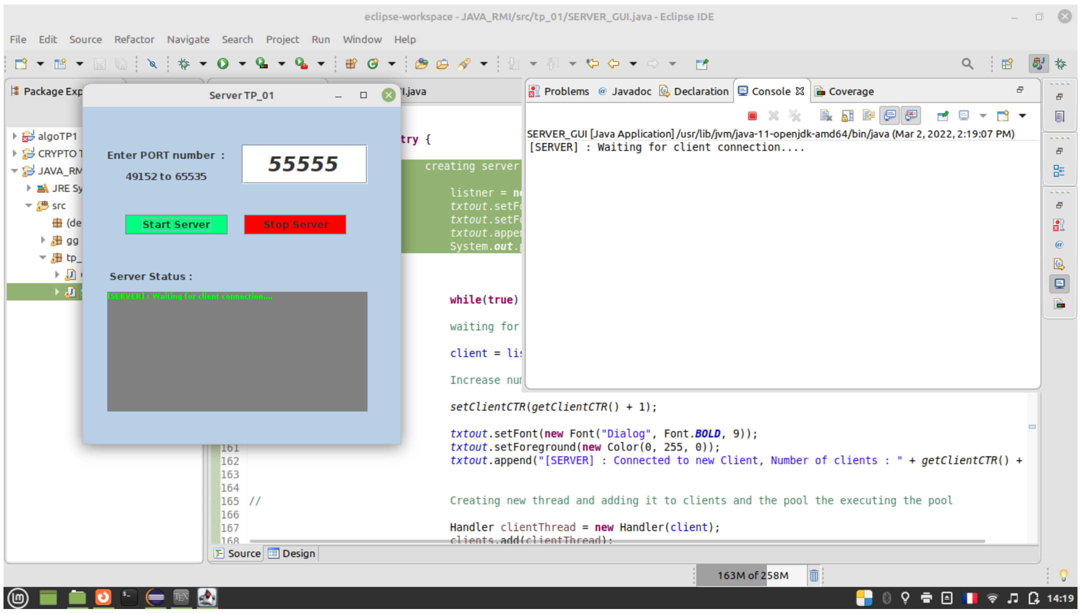
```
CLIENT_GUI [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (Mar 2, 2022, 2:08:18 PM)
[CLIENT] : My address is /127.0.0.1 port 57310
[CLIENT] : The server address is localhost/127.0.0.1 port 55555
[SERVER] : Current Number of Clients = 1
[SERVER] : Recieve new Table
[SERVER] : 1,2,3,4,5,6
[SERVER] : votre tableau ne contient aucun doublon
```

FIGURE 1.6: Showing table.

1.2.4 Le serveur doit afficher :

un message pour indiquer qu'il est en attente de clients

```
//                                  creating server socket

                                         listner = new ServerSocket(PORT);
                                         txtout.setFont(new Font("Dialog",
                                             Font.BOLD, 9));
                                         txtout.setForeground(new Color(0,
                                             255, 0));
                                         txtout.append("[SERVER] : Waiting
                                             for client connection....\n");
                                         System.out.println("[SERVER] :
                                             Waiting for client connection
                                             ....");
```



```
[SERVER] : Waiting for client connection....
```

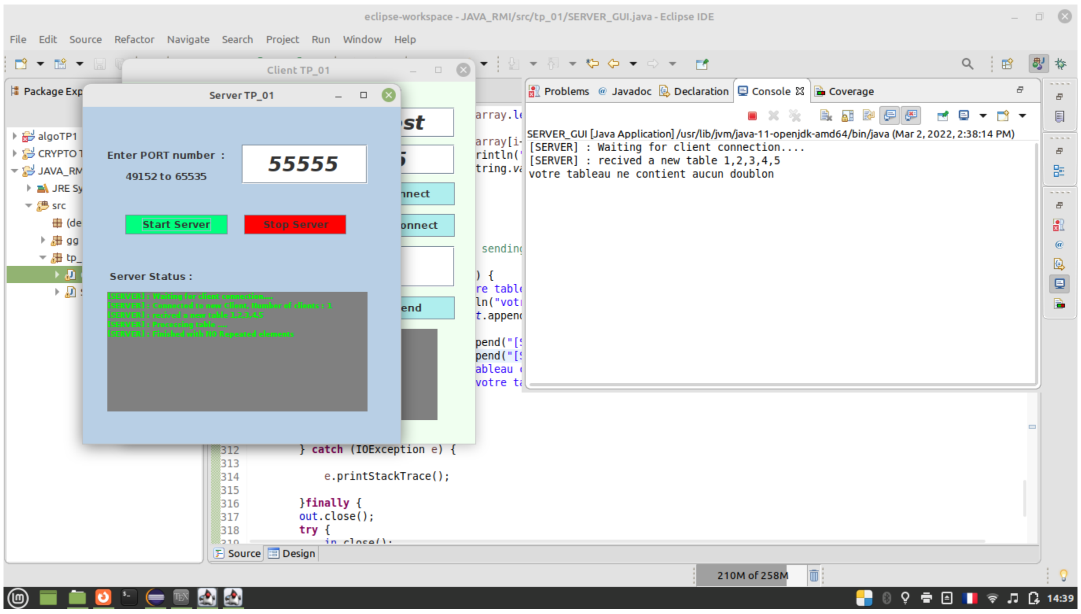
```
setClientCTR(getClientCTR() + 1);

txtout.setFont(new Font("Dialog", Font.BOLD, 9));
txtout.setForeground(new Color(0, 255, 0));
txtout.append("[SERVER] : Connected to new Client, Number of clients : " + getClientCTR() +

//            Creating new thread and adding it to clients and the pool the executing the pool

Handler clientThread = new Handler(client);
clients.add(clientThread);
```

```
SERVER_GUI [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (Mar 2, 2022, 2:19:07 PM)
[SERVER] : Waiting for client connection....
```

FIGURE 1.7: Waiting for clients.

le tableau reçu

```
            SERVER_GUI.txtout.append("[SERVER] : recived a new
                table " + request + "\n");
            System.out.println("[SERVER] : recived a new table
                " + request);
```



```
[SERVER] : Waiting for client connection....
[SERVER] : Connected to new Client, Number of clients : 1
[SERVER] : recived a new table 1,2,3,4,5
[SERVER] : Processing table ....
[SERVER] : Finished with NO Repeated elements
```

```
        } catch (IOException e) {

            e.printStackTrace();

        }finally {
        out.close();
        try {
```

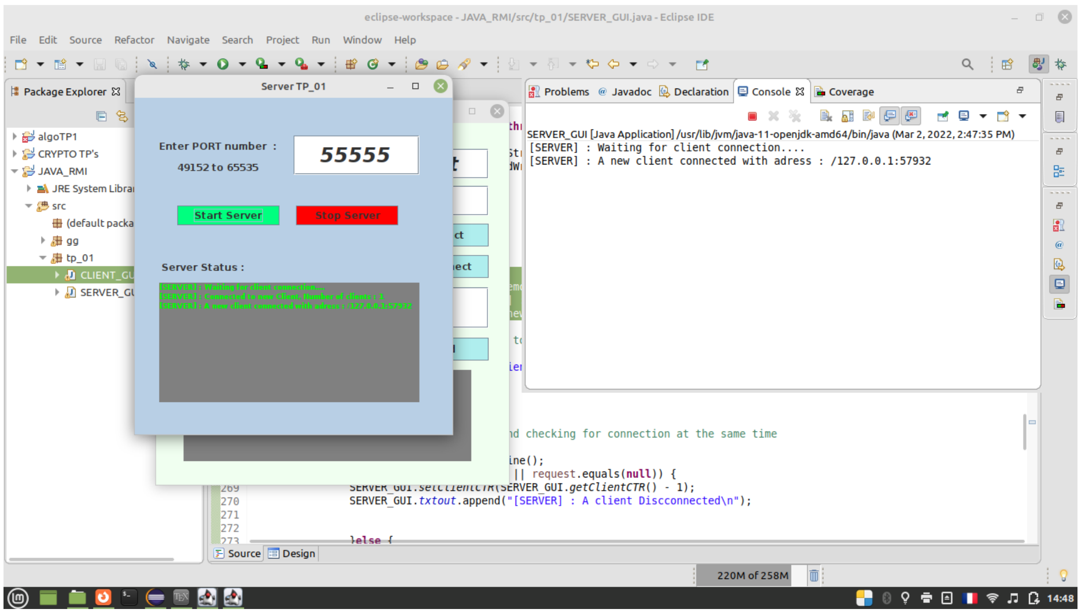
```
SERVER_GUI [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (Mar 2, 2022, 2:38:14 PM)
[SERVER] : Waiting for client connection....
[SERVER] : recived a new table 1,2,3,4,5
votre tableau ne contient aucun doublon
```

FIGURE 1.8: Received table.

l'adresse du client

```
//      new client address
```

```java
        SocketAddress addrC = client.getRemoteSocketAddress();
        SERVER_GUI.txtout.append("[SERVER] : A new client connected
            with adress : " + addrC + "\n");
        System.out.println("[SERVER] : A new client connected with
            adress : " + addrC);
```



Server TP_01
Enter PORT number : 55555
49152 to 65535
Server Status :

```
[SERVER] : Waiting for client connection....
[SERVER] : Connected to new Client, Number of clients : 1
[SERVER] : A new client connected with adress : /127.0.0.1:57932
```

```
SERVER_GUI [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (Mar 2, 2022, 2:47:35 PM)
[SERVER] : Waiting for client connection....
[SERVER] : A new client connected with adress : /127.0.0.1:57932
```

```java
nd checking for connection at the same time
ine();
 || request.equals(null)) {
SERVER_GUI.setClientCTR(SERVER_GUI.getClientCTR() - 1);
SERVER_GUI.txtout.append("[SERVER] : A client Discconnected\n");

}else {
```

FIGURE 1.9: new client connects.

1.3 Question 2

Enrichir le code pour le rendre multi threads.

Why to use threads in network programming?

The reason is simple, we don't want only a single client to connect to server at a particular time but many clients simultaneously. We want our architecture to support multiple clients at the same time. For this reason, we must use threads on server side so that whenever a client request comes, a separate thread can be assigned for handling each request.

Let us take an example, suppose a Date-Time server is located at a place, say X. Being a generic server, it does not serve any particular client, rather to a whole set of generic clients. Also suppose at a particular time, two requests arrives at the server. With our basic server-client program, the request which comes even a nano-second first would be able to connect to the server and the other request would be rejected as no mechanism is provided for handling multiple requests simultaneously. To overcome this problem, we use threading in network programming.

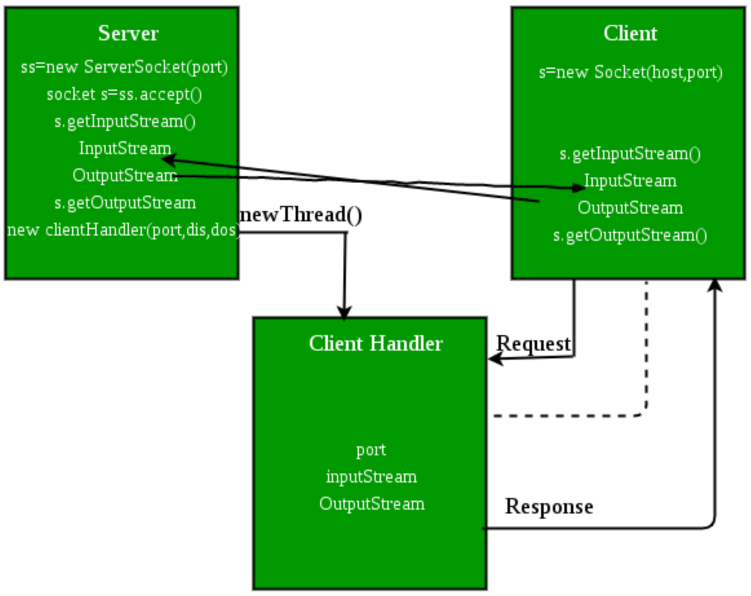


FIGURE 1.10: Server Side Programming.

1.3.1 En plus des taches précédentes, le serveur doit pouvoir :

Traiter plusieurs clients en même temps.

Creating a list to store all connected clients in + an execution pool limited to 10 clients

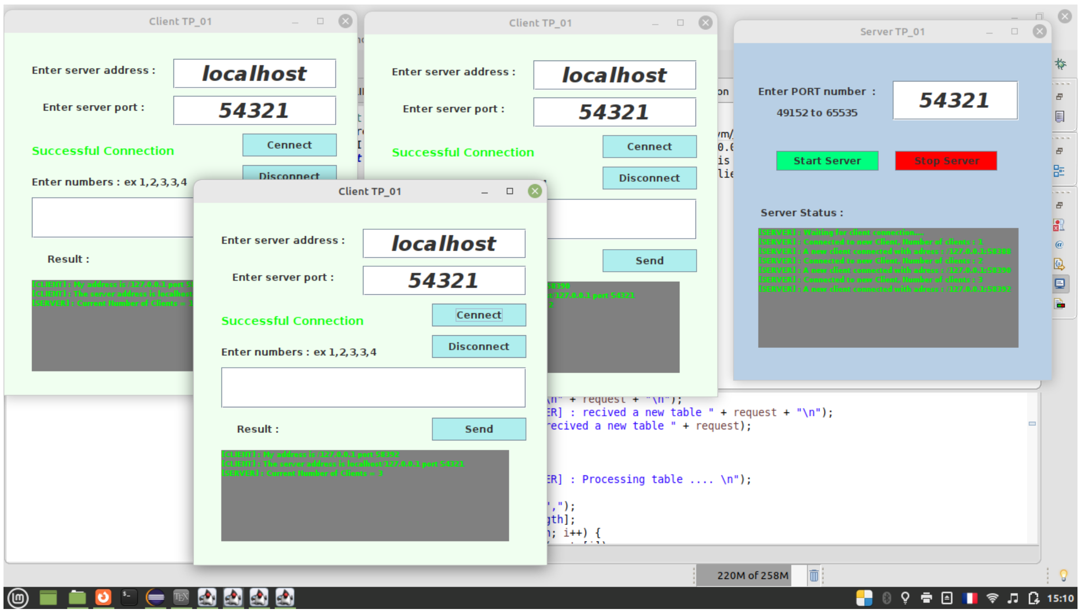
```
//                  creating list to add clients to it

                    ArrayList<Handler> clients = new ArrayList<>();

//                  creating a limited pool of clients of 10

                    ExecutorService pool = Executors.
                        newFixedThreadPool(10);
```

Creating a new thread and sending it to the handler

```
//                                 Creating new thread and adding it
    to clients and the pool the executing the pool

                                   Handler clientThread = new Handler(
                                       client);

                                   clients.add(clientThread);
                                   pool.execute(clientThread);
```

The Handler class and the run method with all what is it doing on every new client sends a table

```
class Handler implements Runnable{

    private Socket client;
    private BufferedReader in;
    private PrintWriter out;

//  constructor for the handler class

    public  Handler(Socket clientSocket) throws IOException {

//      creation of the socket object with the in and out objects
        this.client = clientSocket;
        in = new BufferedReader(new InputStreamReader(client.getInputStream()));
        out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(client.getOutputStream())), true);

    }

    @Override
    public void run() {

//      new client address
        SocketAddress addrC = client.getRemoteSocketAddress();
        SERVER_GUI.txtout.append("[SERVER] : A new client connected with adress : " + addrC + "\n");
        System.out.println("[SERVER] : A new client connected with adress : " + addrC);

//      Sending current Number of clients to the new client

        out.println("Current Number of Clients = " + SERVER_GUI.getClientCTR() + "\n");

        try {
            while (true) {

//          getting a table from client and checking for connection at the same time

                String request = in.readLine();
                if (request.equals("bye") || request.equals(null)) {
                SERVER_GUI.setClientCTR(SERVER_GUI.getClientCTR() - 1);
                SERVER_GUI.txtout.append("[SERVER] : A client Discconnected\n");


                }else {
                out.println("Recieved new Table\n" + request + "\n");
                SERVER_GUI.txtout.append("[SERVER] : recived a new table " + request + "\n");
                System.out.println("[SERVER] : recived a new table " + request);

//              Processing table from client

                SERVER_GUI.txtout.append("[SERVER] : Processing table .... \n");

                String[] parts = request.split(",");
                int[] array = new int[parts.length];
                for (int i = 0; i < parts.length; i++) {
                    array[i] = Integer.parseInt(parts[i]);
                }


//              Sorting the array and finding repeated elements

                Arrays.sort(array);
                ArrayList<String> record = new ArrayList<String>();

                for (int i = 0; i < array.length-1; i++) {

                    if (array[i] == array[i+1]) {
                        System.out.println("duplicate item ["+array[i+1]+"]");
                        record.add(String.valueOf(array[i]));

                    }

                }

//              Checking Results and sending them to client

                if (record.isEmpty()) {
                    out.println("votre tableau ne contient aucun doublon\n");
                    System.out.println("votre tableau ne contient aucun doublon");
                    SERVER_GUI.txtout.append("[SERVER] : Finished with NO Repeated elements \n");
                }else {
                SERVER_GUI.txtout.append("[SERVER] : Finished. Found repeated " + record + "\n");
                SERVER_GUI.txtout.append("[SERVER] : Sending Results to Client\n");
                out.println("votre tableau contient un doublon\n duplicate items are : " + record + "\n");
                System.out.println("votre tableau ne contient aucun doublon" + record);

                }

                }}
        } catch (IOException e) {

                e.printStackTrace();

        }finally {
        out.close();
        try {
                in.close();
        } catch (IOException e) {

                e.printStackTrace();
        }
        try {
                client.close();
        } catch (IOException e) {

                e.printStackTrace();
        }
        }

    }

}
```



```
[SERVER] : Waiting for client connection....
[SERVER] : Connected to new Client, Number of clients : 1
[SERVER] : A new client connected with adress : /127.0.0.1:58388
[SERVER] : Connected to new Client, Number of clients : 2
[SERVER] : A new client connected with adress : /127.0.0.1:58390
[SERVER] : Connected to new Client, Number of clients : 3
[SERVER] : A new client connected with adress : /127.0.0.1:58392
```

```
[CLIENT] : My address is /127.0.0.1 port 58392
[CLIENT] : The server address is localhost/127.0.0.1 port 54321
[SERVER] : Current Number of Clients = 3
```

FIGURE 1.11: Multi clients connecting at the same time.

Afficher le nombre de clients connectés.

adding clients and current number of clients

```
//      new client address
```

```
        SocketAddress addrC = client.getRemoteSocketAddress();
        SERVER_GUI.txtout.append("[SERVER] : A new client connected
            with adress : " + addrC + "\n");
        System.out.println("[SERVER] : A new client connected with
            adress : " + addrC);

//      Sending current Number of clients to the new client

        out.println("Current Number of Clients = " + SERVER_GUI.
            getClientCTR() + "\n");
```

Increasing the clients

```
//                              Increase number of clients

                                setClientCTR(getClientCTR() + 1);
```

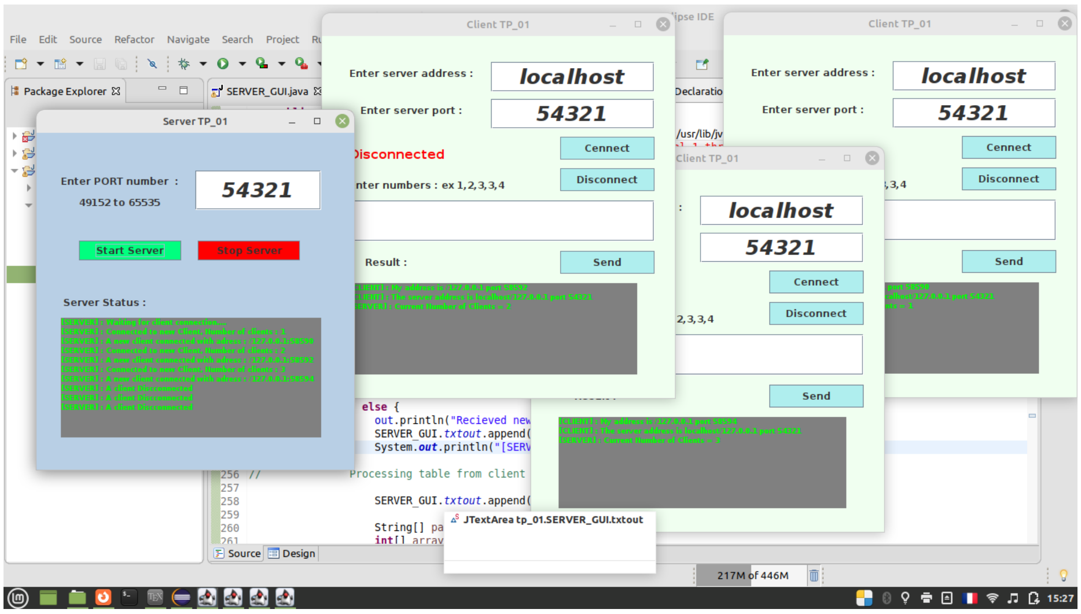
Setters and Getters for the number of clients

```
    public static int getClientCTR() {
        return clientCTR;
    }

    public static void setClientCTR(int clientCTR) {
        SERVER_GUI.clientCTR = clientCTR;
    }
}
```

Client dissconnecting

The client send **bye** message or when the server requests a response and it reacieves a **NULL** response it assumes the client has disconnected.

```
                String request = in.readLine();
                if (request.equals("bye") || request.equals(null))
                    {
                SERVER_GUI.setClientCTR(SERVER_GUI.getClientCTR() -
                     1);
                SERVER_GUI.txtout.append("[SERVER] : A client
                    Dissconnected\n");
                System.out.println("[SERVER] : A client
                    Dissconnected");
```



```
[SERVER] : Waiting for client connection....
[SERVER] : Connected to new Client, Number of clients : 1
[SERVER] : A new client connected with adress : /127.0.0.1:58590
[SERVER] : Connected to new Client, Number of clients : 2
[SERVER] : A new client connected with adress : /127.0.0.1:58592
[SERVER] : Connected to new Client, Number of clients : 3
[SERVER] : A new client connected with adress : /127.0.0.1:58594
[SERVER] : A client Discconnected
[SERVER] : A client Discconnected
[SERVER] : A client Discconnected
```

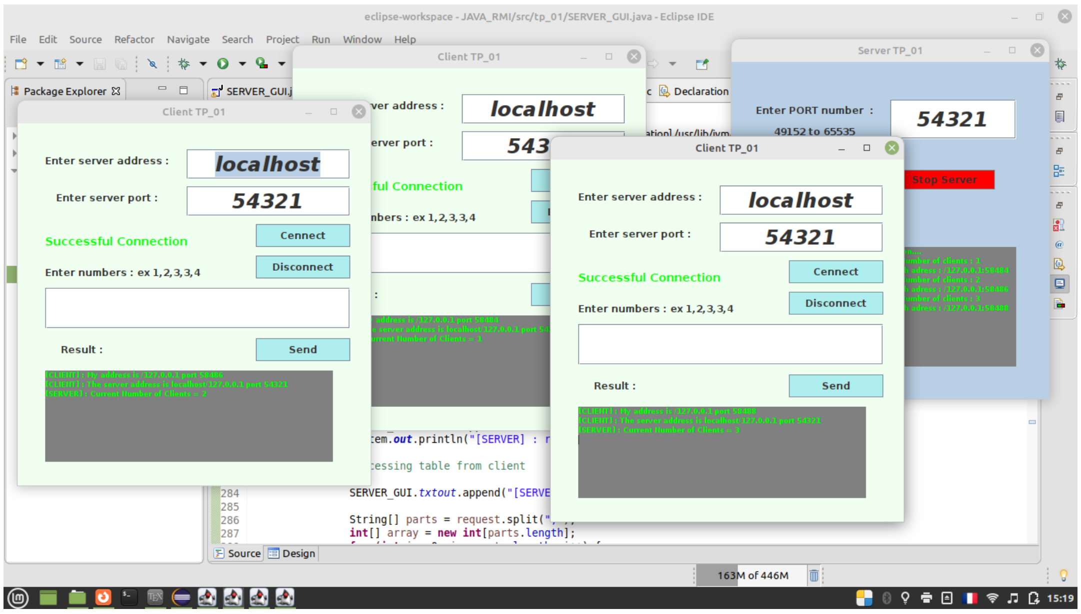
```
[CLIENT] : My address is /127.0.0.1 port 58592
[CLIENT] : The server address is localhost/127.0.0.1 port 54321
[SERVER] : Current Number of Clients = 2
```

```
[CLIENT] : My address is /127.0.0.1 port 58594
[CLIENT] : The server address is localhost/127.0.0.1 port 54321
[SERVER] : Current Number of Clients = 3
```

FIGURE 1.12: Client disconnects.

Transmettre le nombre de clients connectés à chaque nouveau client. Qui doit l'afficher à son tour.

```
//        Sending current Number of clients to the new client

          out.println("Current Number of Clients = " + SERVER_GUI.
              getClientCTR() + "\n");
```



```
[CLIENT] : My address is /127.0.0.1 port 58486
[CLIENT] : The server address is localhost/127.0.0.1 port 54321
[SERVER] : Current Number of Clients = 2
```

```
[CLIENT] : My address is /127.0.0.1 port 58488
[CLIENT] : The server address is localhost/127.0.0.1 port 54321
[SERVER] : Current Number of Clients = 3
```

FIGURE 1.13: Clients receiving client count once connected.

Appendix A

Appendix A

A.1 Java Code for SERVER-GUI.java

```java
package tp_01;


//TP1 Reseaux et Systemes Repartis 2021-2022

//Nom:HADJAZI
//Prenom: Mohammed Hisham
//Specialite:   RSSI      Groupe: 01

//Nom:Ameur
//Prenom: Wassim Malik
//Specialite:   RSSI      Groupe: 01


import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JTextField;
import javax.swing.JLabel;
import javax.swing.SwingConstants;
import javax.swing.SwingUtilities;
import javax.swing.SwingWorker;
import java.awt.Font;
import java.awt.Color;
import javax.swing.JButton;
import javax.swing.UIManager;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketAddress;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.awt.event.ActionEvent;
import javax.swing.JTextArea;
```

```java

public class SERVER_GUI {

    private JFrame frmServerTp;
    private JTextField input;
    private PrintWriter out;
    private BufferedReader in;
    private Socket client;
    private ServerSocket listner;
    static JTextArea txtout;
    private static int clientCTR = 0;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {

        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    SERVER_GUI window = new SERVER_GUI();
                    window.frmServerTp.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the application.
     */
    public SERVER_GUI() {
        initialize();
    }

    /**
     * Initialize the contents of the frame.
     */
    private void initialize() {
        frmServerTp = new JFrame();
        frmServerTp.setBackground(new Color(64, 224, 208));
        frmServerTp.getContentPane().setBackground(UIManager.
            getColor("activeCaption"));
        frmServerTp.setTitle("Server TP_01");
        frmServerTp.setBounds(100, 100, 406, 459);
        frmServerTp.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frmServerTp.getContentPane().setLayout(null);

        input = new JTextField();
        input.setFont(new Font("Dialog", Font.BOLD | Font.ITALIC,
            26));
        input.setHorizontalAlignment(SwingConstants.CENTER);
        input.setBounds(202, 48, 160, 50);
        frmServerTp.getContentPane().add(input);
        input.setColumns(10);

        JLabel lblEnterPortNumber = new JLabel("Enter PORT number
           :");
        lblEnterPortNumber.setBounds(31, 41, 178, 40);
        frmServerTp.getContentPane().add(lblEnterPortNumber);

        JLabel lblTo = new JLabel("49152 to 65535");
        lblTo.setBounds(54, 68, 130, 40);
        frmServerTp.getContentPane().add(lblTo);

        txtout = new JTextArea();
        txtout.setForeground(Color.GREEN);
        txtout.setBackground(Color.GRAY);
        txtout.setBounds(31, 235, 331, 152);
        frmServerTp.getContentPane().add(txtout);

        JLabel lblServerStatus = new JLabel("Server Status :");
        lblServerStatus.setBounds(34, 208, 135, 15);
        frmServerTp.getContentPane().add(lblServerStatus);

        SwingUtilities.invokeLater(new Runnable() {
            public void run() {

                JButton btnStart = new JButton("Start Server");
                btnStart.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent arg0) {

//              getting server port from user

                        int PORT = Integer.parseInt(input.getText()
                           );

//               creating list to add clients to it

                        ArrayList<Handler> clients = new ArrayList
                           <>();

//               creating a limited pool of clients of 10

                        ExecutorService pool = Executors.
                           newFixedThreadPool(10);

                        new SwingWorker() {

                            @Override
                            protected Object doInBackground()
                               throws Exception {

                                try {

//                          creating server socket

                                    listner = new ServerSocket(PORT
                                       );
                                    txtout.setFont(new Font("Dialog
                                       ", Font.BOLD, 9));
                                    txtout.setForeground(new Color
                                       (0, 255, 0));
                                    txtout.append("[SERVER] :
                                        Waiting for client
                                        connection....\n");
                                    System.out.println("[SERVER] :
                                        Waiting for client
                                        connection....");

                                    while (true) {

//                                waiting for clients

                                        client = listner.accept();

//                                Increase number of clients

                                        setClientCTR(getClientCTR()
                                             + 1);

                                        txtout.setFont(new Font("
                                            Dialog", Font.BOLD, 9));
                                        txtout.setForeground(new
                                            Color(0, 255, 0));
                                        txtout.append("[SERVER] :
                                            Connected to new Client,
                                             Number of clients : "
                                                + getClientCTR() +
                                                    "\n");

//                                Creating new thread and adding it
    to clients and the pool the executing the pool

                                        Handler clientThread = new
                                            Handler(client);
                                        clients.add(clientThread);
                                        pool.execute(clientThread);
                                    }
                                } catch (Exception e) {
                                    e.printStackTrace();
                                }

                                return null;
                            }

                        }.execute();

                    }
                });
                btnStart.setBackground(new Color(0, 255, 127));
                btnStart.setBounds(54, 137, 130, 25);
                frmServerTp.getContentPane().add(btnStart);

            }
        });

        JButton btnStop = new JButton("Stop Server");
        btnStop.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
```

```java

                // Close socket input/output stream and client/server socket

                try {
                    out.close();
                    in.close();
                    client.close();
                    listner.close();
                } catch (IOException e) {

                    e.printStackTrace();
                }

            }
        });
        btnStop.setBackground(new Color(255, 0, 0));
        btnStop.setBounds(205, 137, 130, 25);
        frmServerTp.getContentPane().add(btnStop);

        JLabel lblCreatedByHadjazi = new JLabel("Created by HADJAZI + AMOUR , G_01 RSSI");
        lblCreatedByHadjazi.setBounds(77, 401, 293, 15);
        frmServerTp.getContentPane().add(lblCreatedByHadjazi);

    }

    public static int getClientCTR() {
        return clientCTR;
    }

    public static void setClientCTR(int clientCTR) {
        SERVER_GUI.clientCTR = clientCTR;
    }
}

class Handler implements Runnable {

    private Socket client;
    private BufferedReader in;
    private PrintWriter out;

//  constructor for the handler class

    public Handler(Socket clientSocket) throws IOException {

//      creation of the socket object with the in and out objects
        this.client = clientSocket;
        in = new BufferedReader(new InputStreamReader(client.getInputStream()));
        out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(client.getOutputStream())), true);

    }

    @SuppressWarnings("null")
    @Override
    public void run() {

//      new client address
        SocketAddress addrC = client.getRemoteSocketAddress();
        SERVER_GUI.txtout.append("[SERVER] : A new client connected with adress : " + addrC + "\n");
        System.out.println("[SERVER] : A new client connected with adress : " + addrC);

//      Sending current Number of clients to the new client

        out.println("Current Number of Clients = " + SERVER_GUI.getClientCTR() + "\n");

        try {
            while (true) {

//          getting a table from client and checking for connection at the same time

                String request = in.readLine();
                if (request.equals("bye") || request.equals(null)) {
                    SERVER_GUI.setClientCTR(SERVER_GUI.getClientCTR() - 1);
                    SERVER_GUI.txtout.append("[SERVER] : A client Discconnected\n");
                    System.out.println("[SERVER] : A client Discconnected");

                } else {
                    out.println("Recieved new Table\n" + request + "\n");
                    SERVER_GUI.txtout.append("[SERVER] : recived a new table " + request + "\n");
                    System.out.println("[SERVER] : recived a new table " + request);

//              Processing table from client

                    SERVER_GUI.txtout.append("[SERVER] : Processing table .... \n");

                    String[] parts = request.split(",");
                    int[] array = new int[parts.length];
                    for (int i = 0; i < parts.length; i++) {
                        array[i] = Integer.parseInt(parts[i]);
                    }

//              Sorting the array and finding repeated elements

                    Arrays.sort(array);
                    ArrayList<String> record = new ArrayList<String>();

                    for (int i = 0; i < array.length - 1; i++) {

```

```
            if (array[i] == array[i + 1]) {
              System.out.println("duplicate item [" +
                array[i + 1] + "]");
              record.add(String.valueOf(array[i]));

            }

          }

//        Checking Results and sending them to client

          if (record.isEmpty()) {
            out.println("votre tableau ne contient
              aucun doublon\n");
            System.out.println("votre tableau ne
              contient aucun doublon");
            SERVER_GUI.txtout.append("[SERVER] :
              Finished with NO Repeated elements \n");
          } else {
            SERVER_GUI.txtout.append("[SERVER] :
              Finished. Found repeated " + record + "\
              n");
            SERVER_GUI.txtout.append("[SERVER] :
              Sending Results to Client\n");
            out.println("votre tableau contient un
              doublon\n duplicate items are : " +
              record + "\n");
            System.out.println("votre tableau ne
              contient aucun doublon" + record);

          }

        }
      }
    } catch (IOException e) {

      e.printStackTrace();

    } finally {
      out.close();
      try {
        in.close();
      } catch (IOException e) {

        e.printStackTrace();
      }
      try {
        client.close();
      } catch (IOException e) {

        e.printStackTrace();
      }
    }

  }

}
```

A.2 Java Code for CLIENT-GUI.java

```java
package tp_01;


//TP1 Reseaux et Systemes Repartis 2021-2022

//Nom:HADJAZI
//Prenom: Mohammed Hisham
//Specialite:   RSSI      Groupe: 01

//Nom:Ameur
//Prenom: Wassim Malik
//Specialite:   RSSI      Groupe: 01


import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JTextField;
import javax.swing.JLabel;
import javax.swing.SwingConstants;
import javax.swing.SwingWorker;
import javax.swing.JButton;
import java.awt.Font;
import java.awt.Color;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.InetAddress;
import java.net.Socket;
import java.awt.event.ActionEvent;
import javax.swing.JTextArea;

public class CLIENT_GUI {

    private JFrame frmClientTp;
    private JTextField input1;
    private JTextField input2;
    private JTextField input3;
    private PrintWriter out;
    private BufferedReader in;
    private Socket socket;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    CLIENT_GUI window = new CLIENT_GUI();
                    window.frmClientTp.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the application.
     */
    public CLIENT_GUI() {
        initialize();
    }

    /**
     * Initialize the contents of the frame.
     */
    private void initialize() {
        frmClientTp = new JFrame();
        frmClientTp.getContentPane().setBackground(new Color(240,
            255, 240));
        frmClientTp.setBackground(new Color(102, 205, 170));
        frmClientTp.setTitle("Client TP_01");
        frmClientTp.setBounds(100, 100, 451, 491);
        frmClientTp.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frmClientTp.getContentPane().setLayout(null);

        input1 = new JTextField();
        input1.setFont(new Font("Dialog", Font.BOLD | Font.ITALIC,
            26));
        input1.setHorizontalAlignment(SwingConstants.CENTER);
        input1.setBounds(215, 33, 208, 38);
        frmClientTp.getContentPane().add(input1);
        input1.setColumns(10);

        input2 = new JTextField();
        input2.setFont(new Font("Dialog", Font.BOLD | Font.ITALIC,
            26));
        input2.setHorizontalAlignment(SwingConstants.CENTER);
        input2.setColumns(10);
        input2.setBounds(215, 80, 208, 38);
        frmClientTp.getContentPane().add(input2);

        JLabel lblNewLabel = new JLabel("Enter server address :");
        lblNewLabel.setHorizontalAlignment(SwingConstants.CENTER);
        lblNewLabel.setBounds(35, 33, 159, 29);
        frmClientTp.getContentPane().add(lblNewLabel);

        JTextArea txtout = new JTextArea();
        txtout.setForeground(Color.GREEN);
        txtout.setBackground(Color.GRAY);
        txtout.setBounds(35, 314, 366, 116);
        frmClientTp.getContentPane().add(txtout);

        JLabel lblEnterServerPort = new JLabel("Enter server port :
            ");
        lblEnterServerPort.setHorizontalAlignment(SwingConstants.
            CENTER);
        lblEnterServerPort.setBounds(35, 80, 159, 29);
        frmClientTp.getContentPane().add(lblEnterServerPort);

        JLabel label = new JLabel("");
        label.setBounds(35, 130, 208, 40);
        frmClientTp.getContentPane().add(label);

        JButton btnStart = new JButton("Cennect");
        btnStart.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {

                new SwingWorker() {

                    @Override
                    protected Object doInBackground() throws
                       Exception {

                        try {

//              getting user input for server address and port

                            int SERVER_PORT = Integer.parseInt(
                               input2.getText());
                            String SERVER_IP = input1.getText();

//               Establishing socket

                            socket = new Socket(SERVER_IP,
                               SERVER_PORT);
                            out = new PrintWriter(socket.
                               getOutputStream(), true);
                            in = new BufferedReader(new
                               InputStreamReader(socket.
                               getInputStream()));

//               getting Sever and client address and port

                            InetAddress addrS = socket.
                               getInetAddress();
                            InetAddress addrC = socket.
                               getLocalAddress();
                            int portS = socket.getPort();
                            int portC = socket.getLocalPort();

                            txtout.setFont(new Font("Dialog", Font.
                               BOLD, 9));
                            txtout.setForeground(new Color(0, 255,
                               0));
                            txtout.append("[CLIENT] : My address is
                                " + addrC + " port " + portC + "\n"
                               );
                            System.out.println("[CLIENT] : My
                               address is " + addrC + " port " +
                               portC);
                            txtout.append("[CLIENT] : The server
                               address is " + addrS + " port " +
                               portS + "\n");
```

```java
                            System.out.println("[CLIENT] : The server address is " + addrS + " port " + portS);

                            label.setFont(new Font("Dialog", Font.BOLD, 14));
                            label.setForeground(new Color(0, 255, 0));
                            label.setText("Successful Connection");
                        } catch (Exception e) {
                            label.setFont(new Font("Dialog", Font.BOLD, 12));
                            label.setForeground(new Color(255, 0, 0));
                            label.setText("ERROR" + e);
                        }

                        while (true) {

                            String serverResponse = in.readLine();
                            if (serverResponse.isEmpty()) {

                            } else {
                                System.out.println("[SERVER] : " + serverResponse);
                                txtout.setFont(new Font("Dialog", Font.BOLD, 9));
                                txtout.setForeground(new Color(0, 255, 0));
                                txtout.append("[SERVER] : " + serverResponse + "\n");
                            }

                        }

                    }

                }.execute();

            }
        });
        btnStart.setBackground(new Color(175, 238, 238));
        btnStart.setBounds(303, 128, 120, 29);
        frmClientTp.getContentPane().add(btnStart);

        JLabel lblResult = new JLabel("Result :");
        lblResult.setHorizontalAlignment(SwingConstants.CENTER);
        lblResult.setBounds(25, 273, 114, 29);
        frmClientTp.getContentPane().add(lblResult);

        input3 = new JTextField();
        input3.setFont(new Font("Dialog", Font.BOLD, 20));
        input3.setHorizontalAlignment(SwingConstants.CENTER);
        input3.setBounds(35, 209, 388, 52);
        frmClientTp.getContentPane().add(input3);
        input3.setColumns(10);

        JLabel lblEnterNumbers = new JLabel("Enter numbers : ex
            1,2,3,3,4");
        lblEnterNumbers.setBounds(35, 182, 388, 15);
        frmClientTp.getContentPane().add(lblEnterNumbers);

        JButton btnSend = new JButton("Send");
        btnSend.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {

//              getting input from user
                String array = input3.getText();

//              sending input to server
                out.println(array);

//              printing table
                txtout.append("[CLIENT] : the array you provided is
                     : " + array + "\n");

                new SwingWorker() {

                    @Override
                    protected Object doInBackground() throws
                        Exception {

//                      Receiving solution from server

                        try {

                            while (true) {

                                String serverResponse = in.readLine
                                    ();
                                if (serverResponse.isEmpty()) {
                                    // empty response
                                } else {
                                    System.out.println("[SERVER] :
                                        " + serverResponse);
                                    txtout.setFont(new Font("Dialog
                                        ", Font.BOLD, 9));
                                    txtout.setForeground(new Color
                                        (0, 255, 0));
                                    txtout.append("[SERVER] :" +
                                        serverResponse + "\n");
                                }

                            }

                        } catch (Exception e) {
                            e.printStackTrace();
                        }

                        return null;
                    }

                }.execute();

            }
        });
        btnSend.setBackground(new Color(175, 238, 238));
        btnSend.setBounds(303, 273, 120, 29);
        frmClientTp.getContentPane().add(btnSend);

        JButton btnStop = new JButton("Disconnect");
        btnStop.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                out.println("bye");

                try {
                    socket.close();
                } catch (IOException e) {
                }

                label.setFont(new Font("Dialog", Font.BOLD, 16));
                label.setForeground(new Color(255, 0, 0));
                label.setText("Disconnected");

            }
        });
        btnStop.setBackground(new Color(175, 238, 238));
        btnStop.setBounds(303, 168, 120, 29);
        frmClientTp.getContentPane().add(btnStop);

    }
}
```

Bibliography

[1] Pavindu Lakshan. *Fundamentals of Socket Programming in Java*. en. June 2020. URL: https://medium.com/javarevisited/fundamentals-of-socket-programming-in-java-bc9acc30eaf4 (visited on 03/02/2022).

[2] Prafful Mehrotra. *Understanding Java Sockets*. en. Nov. 2019. URL: https://medium.com/swlh/understanding-java-sockets-3fb1f23905c (visited on 03/02/2022).

[3] Manusha Priyanjalee. *Socket Programming in Java*. en. June 2021. URL: https://medium.com/analytics-vidhya/socket-programming-in-java-75918f7e99bf (visited on 03/02/2022).