

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 DEGREE 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

List of Figures	ii
1 Application client-serveur avec les sockets TCP java	1
1.1 Introduction	1
1.1.1 What is a socket, exactly?	1
1.1.2 How do sockets work?	2
1.2 Question 1	3
1.2.1 Si le client envoie 12, 30, 5, 5,100 le serveur répond : »votre tableau contient un doublon «.	4
1.2.2 Si le client envoie 12, 30, 50, 5,100 le serveur répond : »votre tableau ne contient aucun doublon «.	5
1.2.3 Le client doit afficher : son adresse (port et adresse IP) et l'adresse du serveur (port et adresse IP) le tableau envoyé la réponse du serveur	6 6 7 8
1.2.4 Le serveur doit afficher : un message pour indiquer qu'il est en attente de clients le tableau reçu l'adresse du client	9 9 10 10
1.3 Question 2	11
1.3.1 En plus des taches précédentes, le serveur doit pouvoir : Traiter plusieurs clients en même temps. Afficher le nombre de clients connectés. Transmettre le nombre de clients connectés à chaque nouveau client. Qui doit l'afficher à son tour.	12 12 15 17
A Appendix A	18
A.1 Java Code for SERVER-GUI.java	18
A.2 Java Code for CLIENT-GUI.java	25

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.

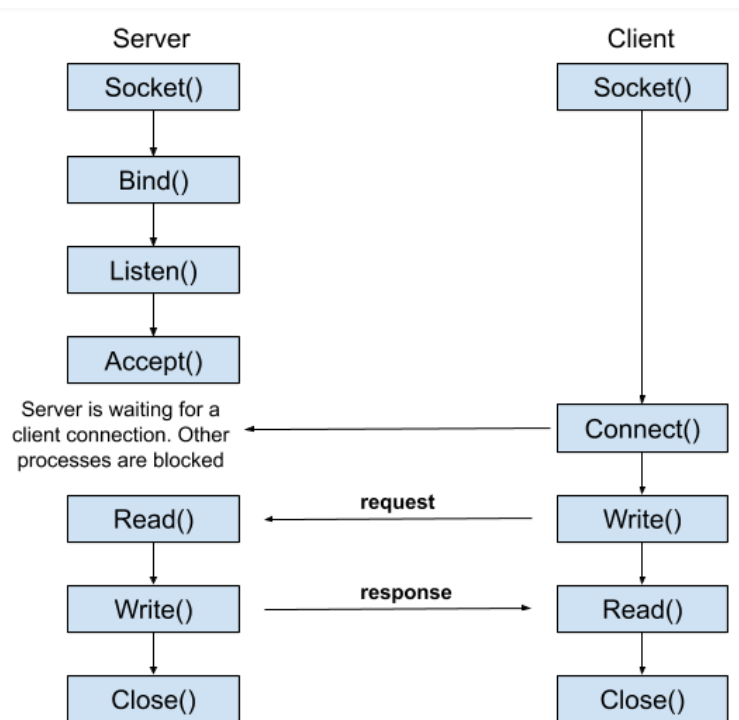


FIGURE 1.1: TCP Socket Communication Flow Diagram.

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

```
1 //          getting input from user
2           String array = input3.getText();
3
4 //          sending input to server
5           out.println(array);
```

Finding Repeated elements algorithm.

```
6           for (int i = 0; i < array.length-1; i++) {
7
8               if (array[i] == array[i+1]) {
9                   System.out.println("duplicate item ["+array
10                      [i+1]+"]");
11                   record.add(String.valueOf(array[i]));
12               }
13
14           }
```

Server sending Positive Response.

```
15 if (record.isEmpty()) {
16     out.println("votre tableau ne contient aucun
17        doublon\n");
18     System.out.println("votre tableau ne contient
19        aucun doublon");
20     SERVER_GUI.txtout.append("[SERVER] : Finished
21        with NO Repeated elements \n");
22 }
```

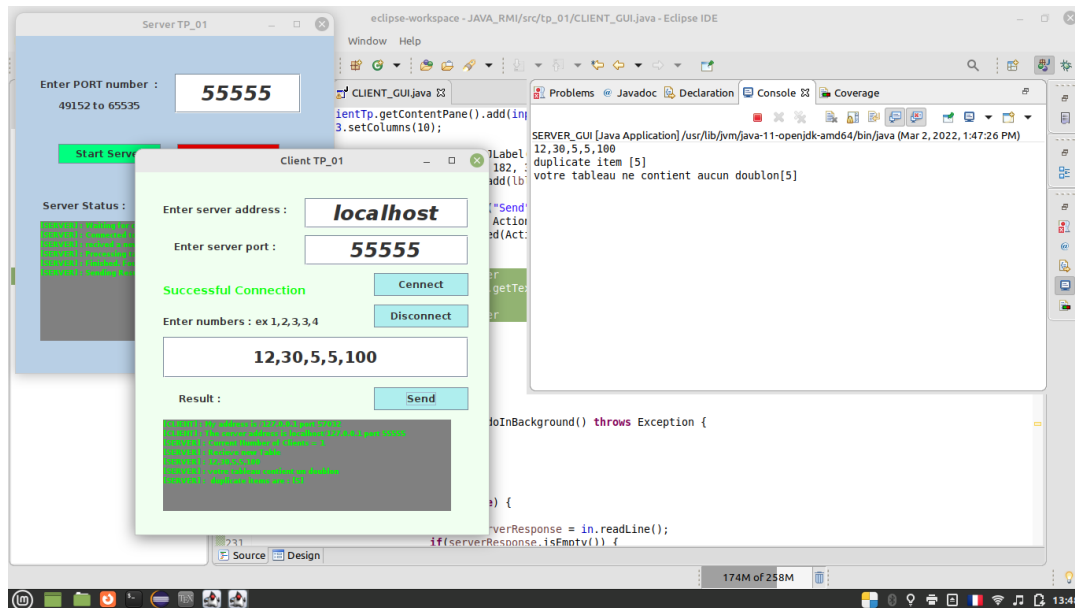


FIGURE 1.2: found duplicates.

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.

```

20 else {
21     SERVER_GUI.txtout.append("[SERVER] : Finished.
22         Found repeated " + record + "\n");
23     SERVER_GUI.txtout.append("[SERVER] : Sending
24         Results to Client\n");
25     out.println("votre tableau contient un doublon\n
26         duplicate items are : " + record + "\n");
27     System.out.println("votre tableau ne contient aucun
28         doublon" + record);
29 }

```

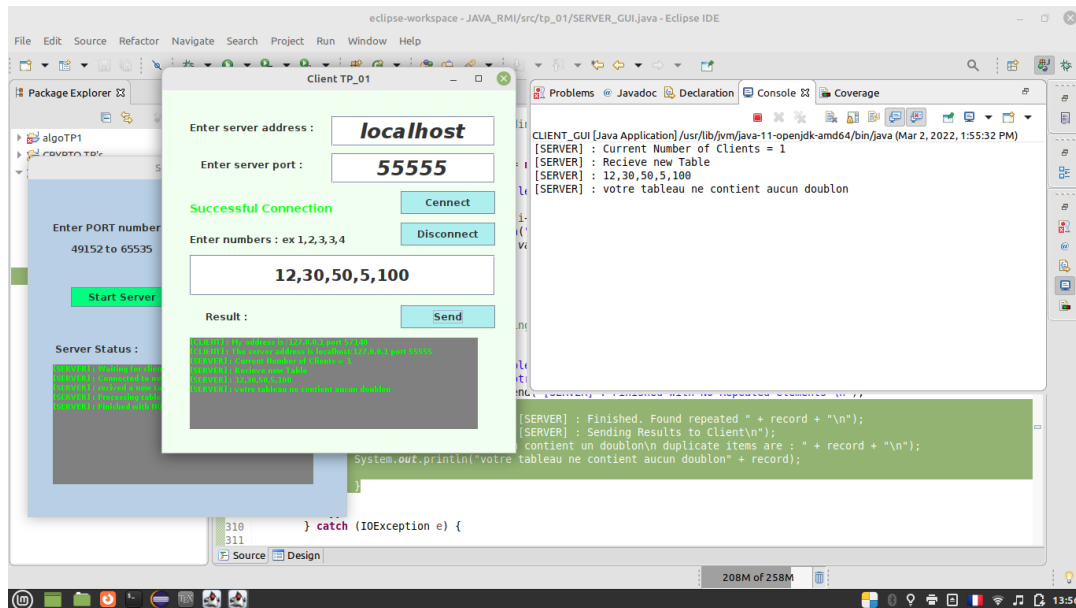


FIGURE 1.3: no duplicates.

1.2.3 Le client doit afficher :

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

```

27 // getting Sever and client address and port
28
29 InetAddress addrS = socket.getInetAddress();
30 InetAddress addrC = socket.getLocalAddress();
31 int portS = socket.getPort();
32 int portC = socket.getLocalPort();
33
34 txtout.setFont(new Font("Dialog", Font.BOLD, 9));
35 txtout.setForeground(new Color(0, 255, 0));
36 txtout.append("[CLIENT] : My address is " + addrC
37 + " port " + portC + "\n");
38 System.out.println("[CLIENT] : My address is " +
39 addrC + " port " + portC);
40 txtout.append("[CLIENT] : The server address is "
41 + addrS + " port " + portS + "\n");
42 System.out.println("[CLIENT] : The server address
43 is " + addrS + " port " + portS);

```

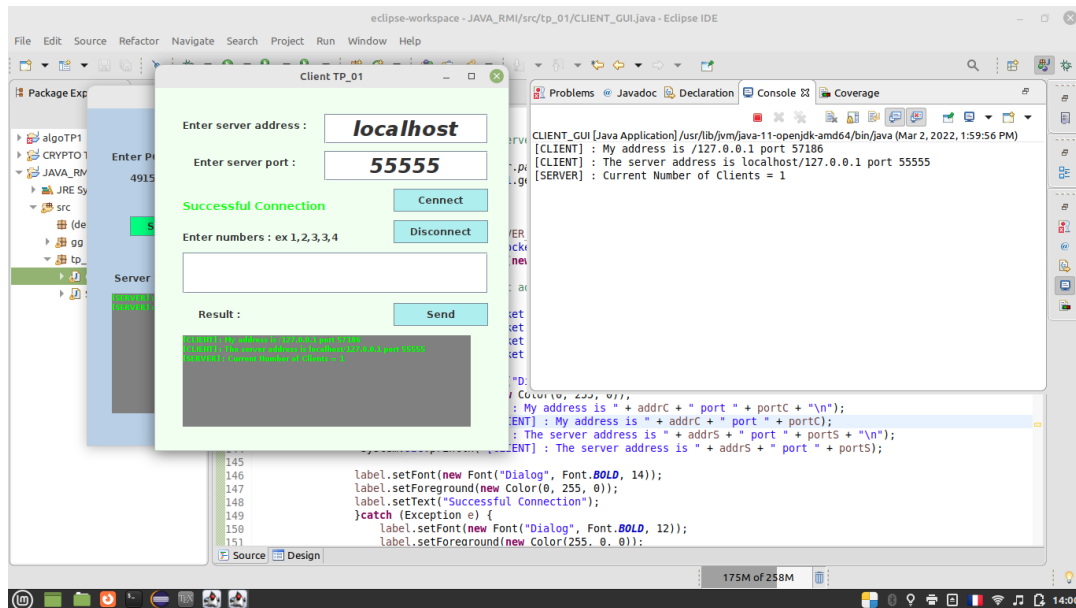



FIGURE 1.4: Showing client/server IP addresses and ports.

le tableau envoyé

```

40 //          getting input from user
41          String array = input3.getText();
42
43 //          sending input to server
44          out.println(array);
45
46 //          printing table
47          txtout.append("[CLIENT] : the array you provided is
          : " + array + "\n");

```

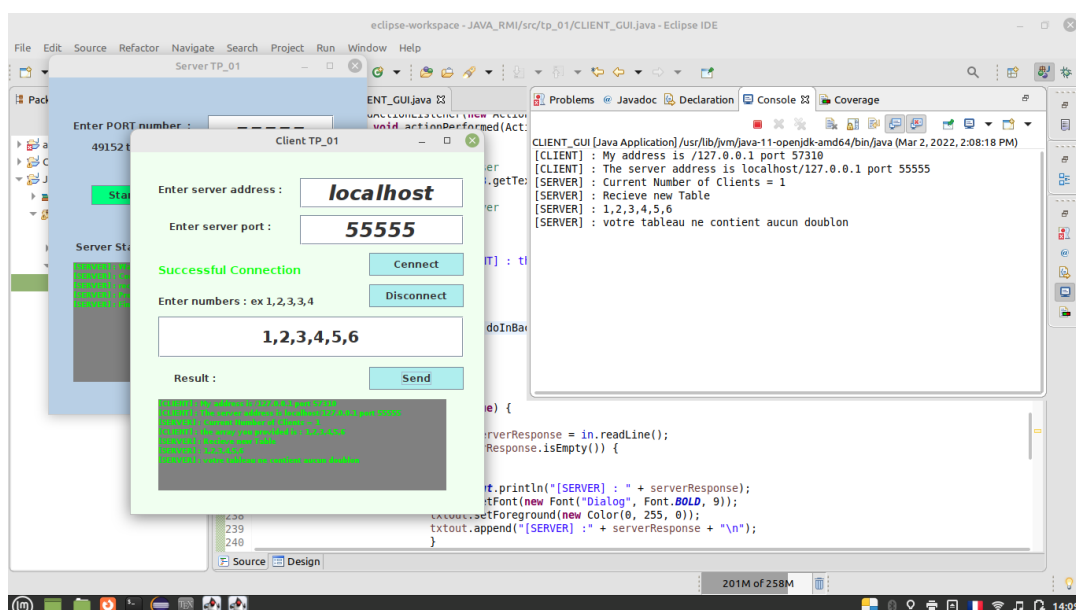


FIGURE 1.5: Showing table.

la réponse du serveur

```
48 //                      Receiving solution from server
49
50     try {
51
52
53
54         while(true) {
55
56             String serverResponse = in.readLine();
57             if(serverResponse.isEmpty()) {
58                 // empty response
59             }else {
60                 System.out.println("[SERVER] : " +
61                     serverResponse);
62                 txtout.setFont(new Font("Dialog", Font.
63                     BOLD, 9));
64                 txtout.setForeground(new Color(0, 255,
65                     0));
66                 txtout.append("[SERVER] :" +
67                     serverResponse + "\n");
68             }
69
70         }
71
72     }catch(Exception e) {
73         e.printStackTrace();
74     }
```

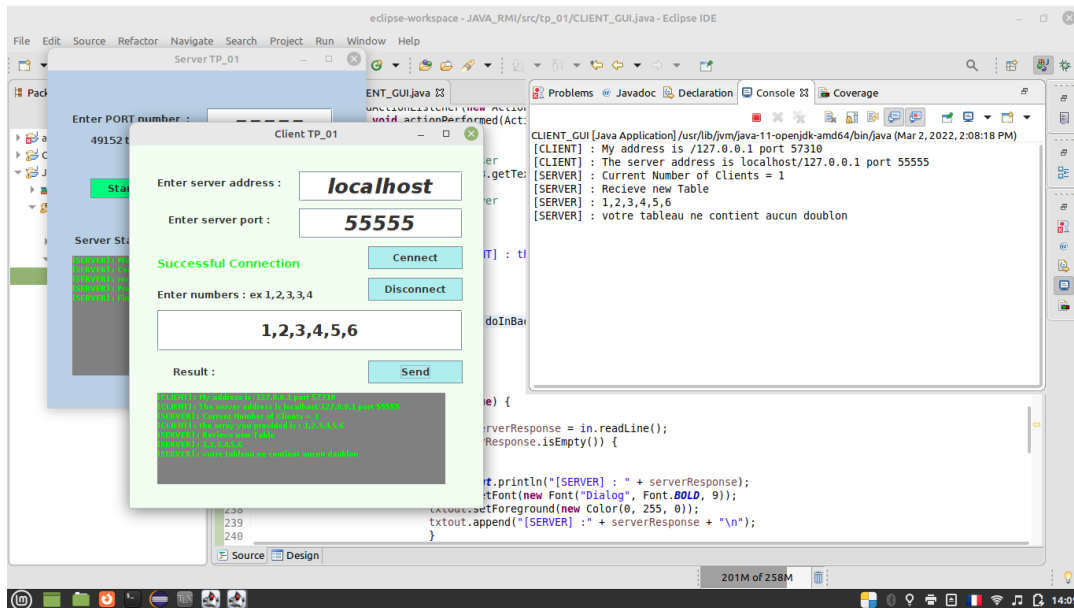


FIGURE 1.6: Showing table.

1.2.4 Le serveur doit afficher :

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

```

73 //                                creating server socket
74
75 listner = new ServerSocket(PORT);
76 txtout.setFont(new Font("Dialog",
77                               Font.BOLD, 9));
78 txtout.setForeground(new Color(0,
79                               255, 0));
80 txtout.append("[SERVER] : Waiting
81               for client connection....\n");
82 System.out.println("[SERVER] :
83               Waiting for client connection
84               ....");

```

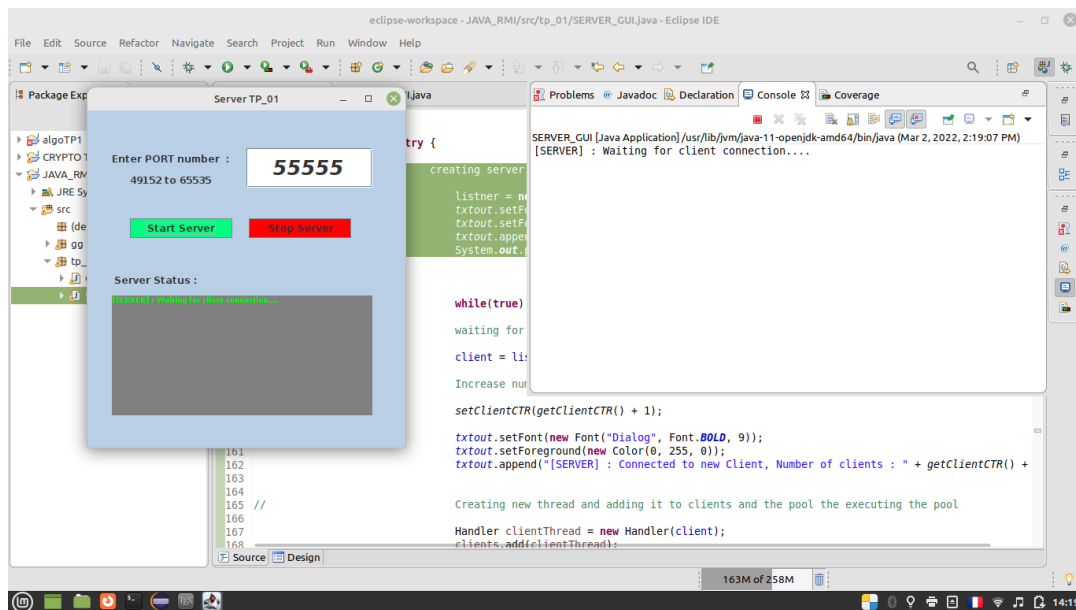


FIGURE 1.7: Waiting for clients.

le tableau reçu

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

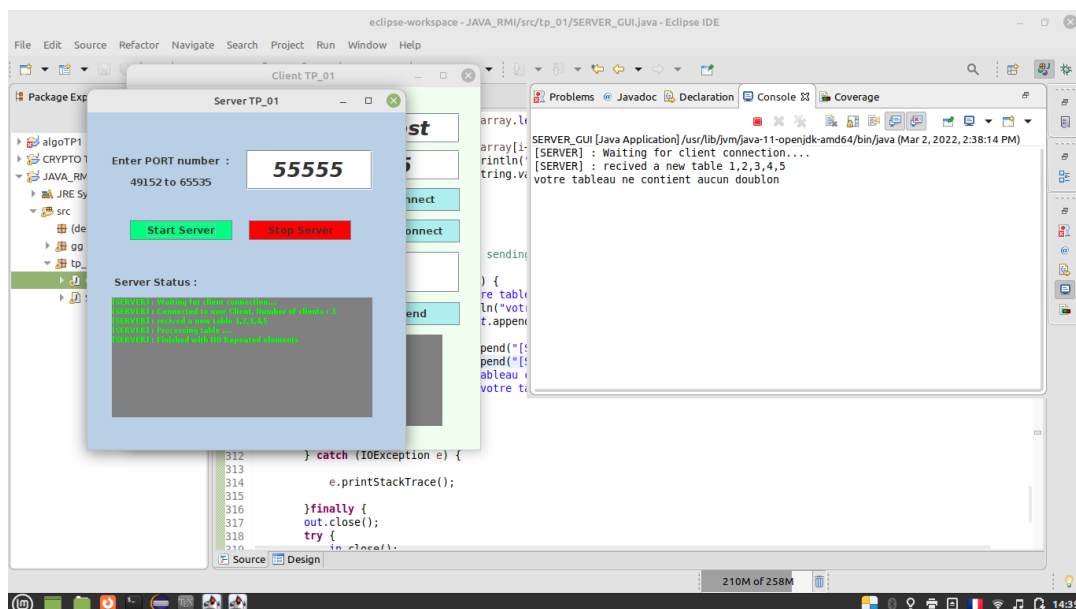


FIGURE 1.8: Received table.

l'adresse du client

```
82 //      new client address
```

```

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

```

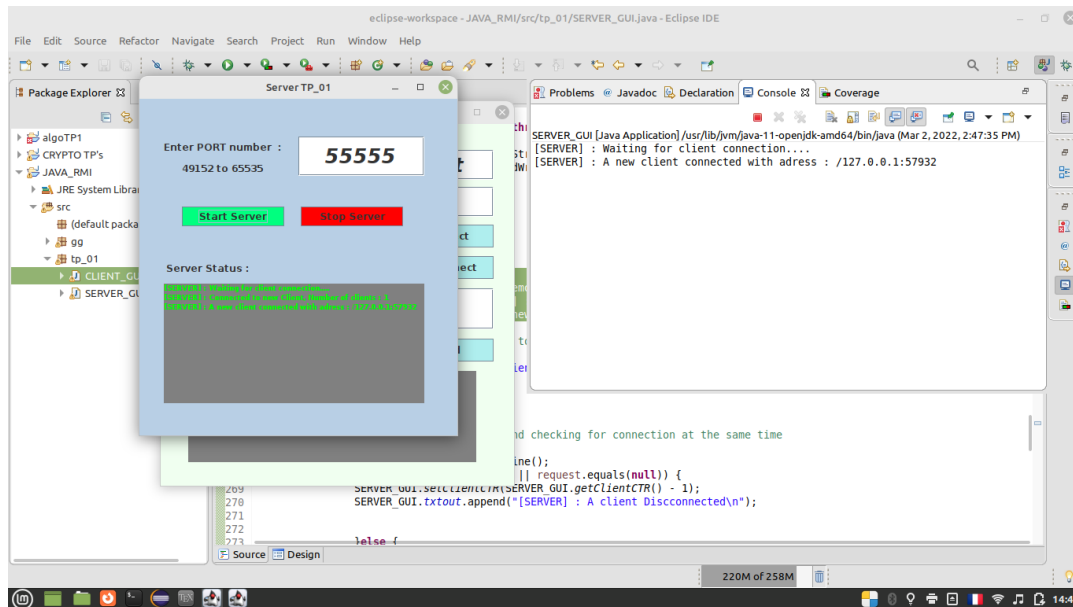


FIGURE 1.9: new client connects.

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.

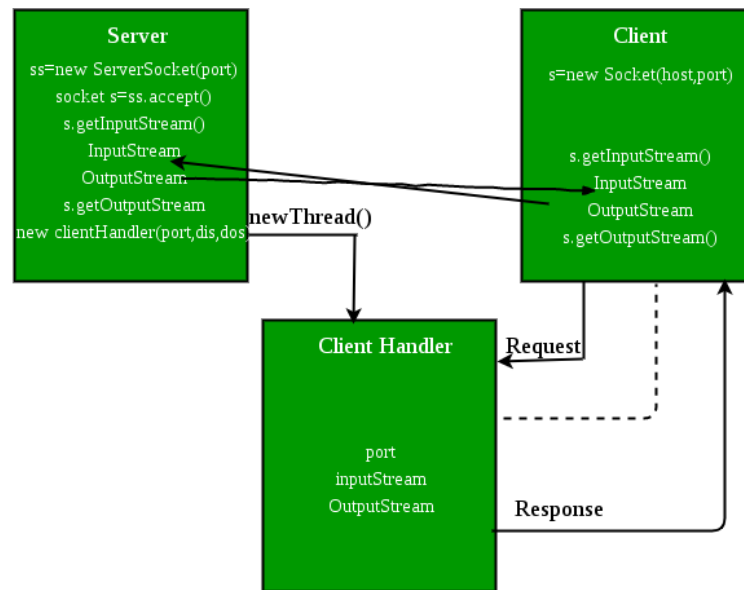


FIGURE 1.10: Server Side Programming.

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

```

86 //          creating list to add clients to it
87
88     ArrayList<Handler> clients = new ArrayList<>();
89
90 //          creating a limited pool of clients of 10
91
92     ExecutorService pool = Executors.
        newFixedThreadPool(10);

```

Creating a new thread and sending it to the handler

```

93 //          Creating new thread and adding it
          to clients and the pool the executing the pool
94
95     Handler clientThread = new Handler(
        client);
96
97     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

```

98 class Handler implements Runnable{
99
100     private Socket client;

```

```
101     private BufferedReader in;
102     private PrintWriter out;
103
104     // constructor for the handler class
105
106     public Handler(Socket clientSocket) throws IOException {
107
108         // creation of the socket object with the in and out objects
109         this.client = clientSocket;
110         in = new BufferedReader(new InputStreamReader(client.
111             getInputStream()));
112         out = new PrintWriter(new BufferedWriter(new
113             OutputStreamWriter(client.getOutputStream()), true);
114     }
115
116     @Override
117     public void run() {
118
119         // new client address
120         SocketAddress addrC = client.getRemoteSocketAddress();
121         SERVER_GUI.txtout.append("[SERVER] : A new client connected
122             with address : " + addrC + "\n");
123         System.out.println("[SERVER] : A new client connected with
124             adress : " + addrC);
125
126         // Sending current Number of clients to the new client
127         out.println("Current Number of Clients = " + SERVER_GUI.
128             getClientCTR() + "\n");
129
130         try {
131             while (true) {
132
133                 // getting a table from client and checking for connection
134                 // at the same time
135
136                 String request = in.readLine();
137                 if (request.equals("bye") || request.equals(null))
138                 {
139                     SERVER_GUI.setClientCTR(SERVER_GUI.getClientCTR() -
140                         1);
141                     SERVER_GUI.txtout.append("[SERVER] : A client
142                         Disconnected\n");
143
144                 }else {
145                     out.println("Recieved new Table\n" + request + "\n"
146                         );
147                     SERVER_GUI.txtout.append("[SERVER] : recived a new
148                         table " + request + "\n");
```

```
141         System.out.println("[SERVER] : recived a new table  
142             " + request);  
143 //         Processing table from client  
144  
145         SERVER_GUI.txtout.append("[SERVER] : Processing  
146             table .... \n");  
147  
148         String[] parts = request.split(",");  
149         int[] array = new int[parts.length];  
150         for (int i = 0; i < parts.length; i++) {  
151             array[i] = Integer.parseInt(parts[i]);  
152         }  
153  
154 //         Sorting the array and finding repeated elements  
155  
156         Arrays.sort(array);  
157         ArrayList<String> record = new ArrayList<String>();  
158  
159         for (int i = 0; i < array.length-1; i++) {  
160  
161             if (array[i] == array[i+1]) {  
162                 System.out.println("duplicate item [" + array  
163                     [i+1] + "]");  
164                 record.add(String.valueOf(array[i]));  
165             }  
166  
167         }  
168  
169 //         Checking Results and sending them to client  
170  
171         if (record.isEmpty()) {  
172             out.println("votre tableau ne contient aucun  
173                 doublon\n");  
174             System.out.println("votre tableau ne contient  
175                 aucun doublon");  
176             SERVER_GUI.txtout.append("[SERVER] : Finished  
177                 with NO Repeated elements \n");  
178         }else {  
179             SERVER_GUI.txtout.append("[SERVER] : Finished.  
180                 Found repeated " + record + "\n");  
181             SERVER_GUI.txtout.append("[SERVER] : Sending  
182                 Results to Client\n");  
183             out.println("votre tableau contient un doublon\n  
184                 duplicate items are : " + record + "\n");  
185             System.out.println("votre tableau ne contient aucun  
186                 doublon" + record);  
187         }  
188     }  
189 }
```



```

182
183     }}
184 } catch (IOException e) {
185
186     e.printStackTrace();
187
188 }finally {
189 out.close();
190 try {
191     in.close();
192 } catch (IOException e) {
193
194     e.printStackTrace();
195 }
196 try {
197     client.close();
198 } catch (IOException e) {
199
200     e.printStackTrace();
201 }
202 }
203
204 }
205
206 }

```

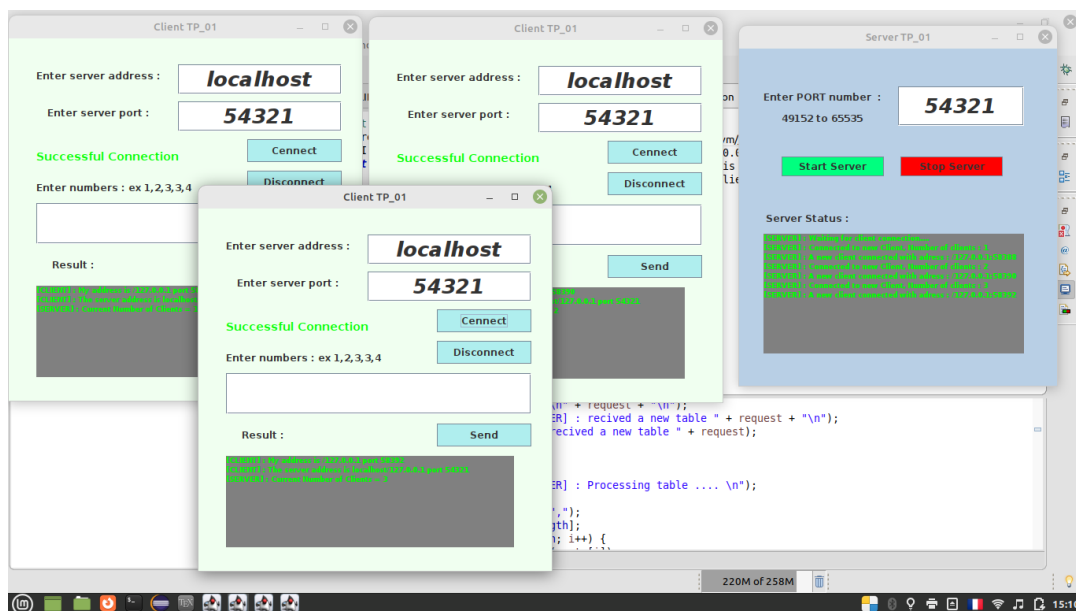


FIGURE 1.11: Multi clients connecting at the same time.

Afficher le nombre de clients connectés.

adding clients and current number of clients

```

207 //      new client address

```

```

208     SocketAddress addrC = client.getRemoteSocketAddress();
209     SERVER_GUI.txtout.append("[SERVER] : A new client connected
        with address : " + addrC + "\n");
210     System.out.println("[SERVER] : A new client connected with
        address : " + addrC);
211
212     //      Sending current Number of clients to the new client
213
214     out.println("Current Number of Clients = " + SERVER_GUI.
        getClientCTR() + "\n");

```

Increasing the clients

```

215     //      Increase number of clients
216
217     setClientCTR(getClientCTR() + 1);

```

Setters and Getters for the number of clients

```

218     public static int getClientCTR() {
219         return clientCTR;
220     }
221
222     public static void setClientCTR(int clientCTR) {
223         SERVER_GUI.clientCTR = clientCTR;
224     }
225 }

```

Client disconnecting

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

```

226     String request = in.readLine();
227     if (request.equals("bye") || request.equals(null))
228     {
229         SERVER_GUI.setClientCTR(SERVER_GUI.getClientCTR() -
            1);
230         SERVER_GUI.txtout.append("[SERVER] : A client
            Disconnected\n");
231         System.out.println("[SERVER] : A client
            Disconnected");

```

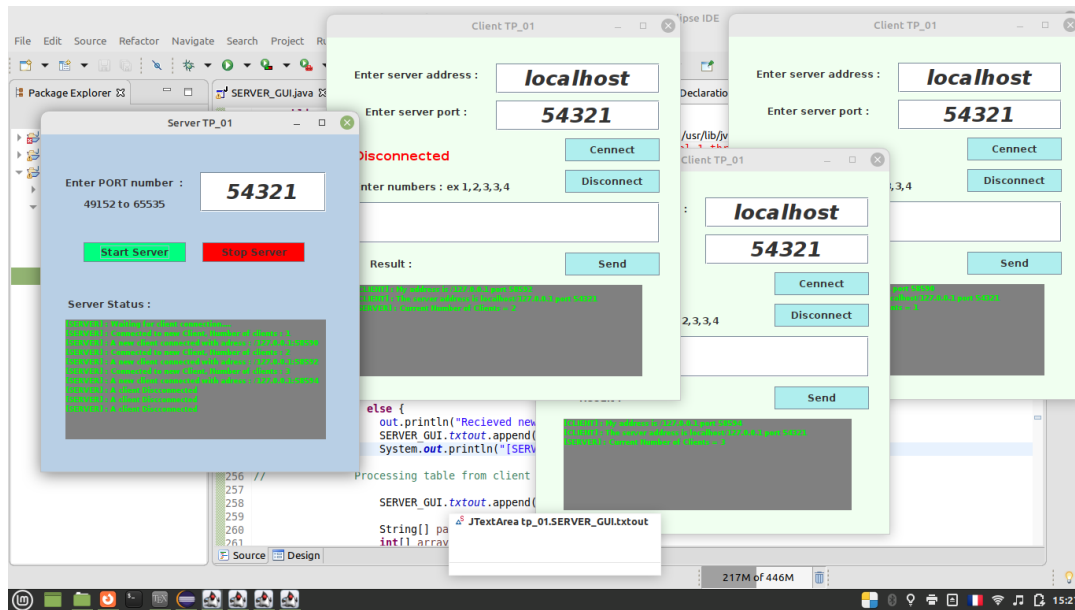


FIGURE 1.12: Client disconnects.

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

```

231 //      Sending current Number of clients to the new client
232
233      out.println("Current Number of Clients = " + SERVER_GUI.
          getClientCTR() + "\n");

```

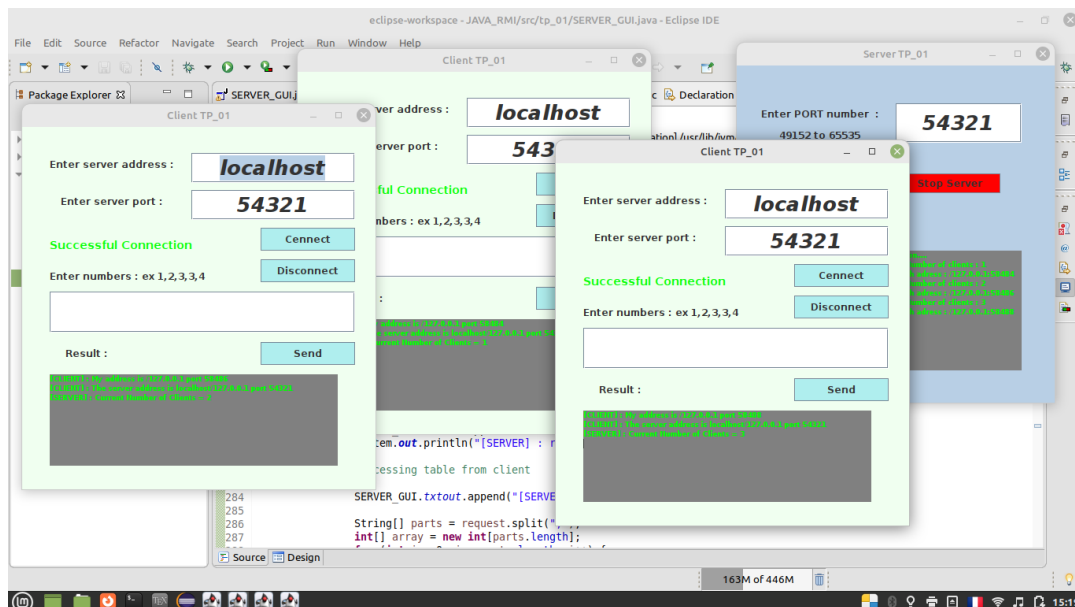


FIGURE 1.13: Clients receiving client count once connected.

Appendix A

Appendix A

A.1 Java Code for SERVER-GUI.java

```

234 package tp_01;
235
236
237 //TP1 Réseaux et Systèmes Répartis 2021-2022
238
239 //Nom:HADJAZI
240 //Prenom: Mohammed Hisham
241 //Specialite:  RSSI      Groupe: 01
242
243 //Nom:Ameur
244 //Prenom: Wassim Malik
245 //Specialite:  RSSI      Groupe: 01
246
247
248 import java.awt.EventQueue;
249 import javax.swing.JFrame;
250 import javax.swing.JTextField;
251 import javax.swing.JLabel;
252 import javax.swing.SwingConstants;
253 import javax.swing.SwingUtilities;
254 import javax.swing.SwingWorker;
255 import java.awt.Font;
256 import java.awt.Color;
257 import javax.swing.JButton;
258 import javax.swing.UIManager;
259 import java.awt.event.ActionListener;
260 import java.io.BufferedReader;
261 import java.io.BufferedWriter;
262 import java.io.IOException;
263 import java.io.InputStreamReader;
264 import java.io.OutputStreamWriter;
265 import java.io.PrintWriter;
266 import java.net.InetAddress;
267 import java.net.ServerSocket;
268 import java.net.Socket;
269 import java.net.SocketAddress;
270 import java.util.ArrayList;
271 import java.util.Arrays;
272 import java.util.concurrent.ExecutorService;
273 import java.util.concurrent.Executors;
274 import java.awt.event.ActionEvent;
275 import javax.swing.JTextArea;

```

```

276
277 public class SERVER_GUI {
278
279     private JFrame frmServerTp;
280     private JTextField input;
281     private PrintWriter out;
282     private BufferedReader in;
283     private Socket client;
284     private ServerSocket listner;
285     static JTextArea txtout;
286     private static int clientCTR = 0;
287
288     /**
289      * Launch the application.
290      */
291     public static void main(String[] args) {
292
293         EventQueue.invokeLater(new Runnable() {
294             public void run() {
295                 try {
296                     SERVER_GUI window = new SERVER_GUI();
297                     window.frmServerTp.setVisible(true);
298                 } catch (Exception e) {
299                     e.printStackTrace();
300                 }
301             }
302         });
303     }
304
305     /**
306      * Create the application.
307      */
308     public SERVER_GUI() {
309         initialize();
310     }
311
312     /**
313      * Initialize the contents of the frame.
314      */
315     private void initialize() {
316         frmServerTp = new JFrame();
317         frmServerTp.setBackground(new Color(64, 224, 208));
318         frmServerTp.getContentPane().setBackground(UIManager.
319             getColor("activeCaption"));
320         frmServerTp.setTitle("Server TP_01");
321         frmServerTp.setBounds(100, 100, 406, 459);
322         frmServerTp.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
323         frmServerTp.getContentPane().setLayout(null);
324
325         input = new JTextField();
326         input.setFont(new Font("Dialog", Font.BOLD | Font.ITALIC,
327             26));
328         input.setHorizontalAlignment(SwingConstants.CENTER);
329         input.setBounds(202, 48, 160, 50);
330         frmServerTp.getContentPane().add(input);
331         input.setColumns(10);

```

```

331     JLabel lblEnterPortNumber = new JLabel("Enter PORT number
332         :");
333     lblEnterPortNumber.setBounds(31, 41, 178, 40);
334     frmServerTp.getContentPane().add(lblEnterPortNumber);
335
336     JLabel lblTo = new JLabel("49152 to 65535");
337     lblTo.setBounds(54, 68, 130, 40);
338     frmServerTp.getContentPane().add(lblTo);
339
340     JTextArea txtout = new JTextArea();
341     txtout.setForeground(Color.GREEN);
342     txtout.setBackground(Color.GRAY);
343     txtout.setBounds(31, 235, 331, 152);
344     frmServerTp.getContentPane().add(txtout);
345
346     JLabel lblServerStatus = new JLabel("Server Status :");
347     lblServerStatus.setBounds(34, 208, 135, 15);
348     frmServerTp.getContentPane().add(lblServerStatus);
349
350     SwingUtilities.invokeLater(new Runnable() {
351         public void run() {
352
353             JButton btnStart = new JButton("Start Server");
354             btnStart.addActionListener(new ActionListener() {
355                 public void actionPerformed(ActionEvent arg0) {
356
357                 // getting server port from user
358
359                     int PORT = Integer.parseInt(input.getText()
360                         );
361
362                 // creating list to add clients to it
363
364                     ArrayList<Handler> clients = new ArrayList
365                         <>();
366
367                 // creating a limited pool of clients of 10
368
369                     ExecutorService pool = Executors.
370                         newFixedThreadPool(10);
371
372                     new SwingWorker() {
373
374                         @Override
375                         protected Object doInBackground()
376                             throws Exception {
377
378                             try {
379
380                             // creating server socket
381
382                                 listner = new ServerSocket(PORT
383                                     );
384                                 txtout.setFont(new Font("Dialog
385                                     ", Font.BOLD, 9));
386                                 txtout.setForeground(new Color
387                                     (0, 255, 0));

```

```

380         txtout.append("[SERVER] :
           Waiting for client
           connection....\n");
381         System.out.println("[SERVER] :
           Waiting for client
           connection....");
382
383         while (true) {
384
385             //           waiting for clients
386
387             client = listner.accept();
388
389             //           Increase number of clients
390
391             setClientCTR(getClientCTR()
                           + 1);
392
393             txtout.setFont(new Font("
           Dialog", Font.BOLD, 9));
394             txtout.setForeground(new
           Color(0, 255, 0));
395             txtout.append("[SERVER] :
           Connected to new Client,
           Number of clients : "
                           + getClientCTR() +
                           "\n");
396
397             //           Creating new thread and adding it
           to clients and the pool the executing the pool
398
399
400             Handler clientThread = new
           Handler(client);
401             clients.add(clientThread);
402             pool.execute(clientThread);
403         }
404         } catch (Exception e) {
405             e.printStackTrace();
406         }
407
408         return null;
409     }
410
411     }.execute();
412
413     }
414     });
415     btnStart.setBackground(new Color(0, 255, 127));
416     btnStart.setBounds(54, 137, 130, 25);
417     frmServerTp.getContentPane().add(btnStart);
418
419     }
420     });
421
422     JButton btnStop = new JButton("Stop Server");
423     btnStop.addActionListener(new ActionListener() {
424         public void actionPerformed(ActionEvent arg0) {

```

```

425
426         // Close socket input/output stream and client/
         server socket
427
428         try {
429             out.close();
430             in.close();
431             client.close();
432             listner.close();
433         } catch (IOException e) {
434
435             e.printStackTrace();
436         }
437
438     }
439 });
440 btnStop.setBackground(new Color(255, 0, 0));
441 btnStop.setBounds(205, 137, 130, 25);
442 frmServerTp.getContentPane().add(btnStop);
443
444 JLabel lblCreatedByHadjazi = new JLabel("Created by HADJAZI
         + AMOUR , G_01 RSSI");
445 lblCreatedByHadjazi.setBounds(77, 401, 293, 15);
446 frmServerTp.getContentPane().add(lblCreatedByHadjazi);
447
448 }
449
450 public static int getClientCTR() {
451     return clientCTR;
452 }
453
454 public static void setClientCTR(int clientCTR) {
455     SERVER_GUI.clientCTR = clientCTR;
456 }
457 }
458
459 class Handler implements Runnable {
460
461     private Socket client;
462     private BufferedReader in;
463     private PrintWriter out;
464
465     // constructor for the handler class
466
467     public Handler(Socket clientSocket) throws IOException {
468
469         // creation of the socket object with the in and out objects
470         this.client = clientSocket;
471         in = new BufferedReader(new InputStreamReader(client.
             getInputStream()));
472         out = new PrintWriter(new BufferedWriter(new
             OutputStreamWriter(client.getOutputStream()), true);
473
474     }
475
476     @SuppressWarnings("null")
477     @Override

```



```

478     public void run() {
479
480         //      new client address
481         SocketAddress addrC = client.getRemoteSocketAddress();
482         SERVER_GUI.txtout.append("[SERVER] : A new client connected
           with address : " + addrC + "\n");
483         System.out.println("[SERVER] : A new client connected with
           address : " + addrC);
484
485         //      Sending current Number of clients to the new client
486
487         out.println("Current Number of Clients = " + SERVER_GUI.
           getClientCTR() + "\n");
488
489         try {
490             while (true) {
491
492                 //      getting a table from client and checking for connection
           at the same time
493
494                 String request = in.readLine();
495                 if (request.equals("bye") || request.equals(null))
496                 {
497                     SERVER_GUI.setClientCTR(SERVER_GUI.getClientCTR
           () - 1);
498                     SERVER_GUI.txtout.append("[SERVER] : A client
           Disconnected\n");
499                     System.out.println("[SERVER] : A client
           Disconnected");
500                 } else {
501                     out.println("Recieved new Table\n" + request +
           "\n");
502                     SERVER_GUI.txtout.append("[SERVER] : recived a
           new table " + request + "\n");
503                     System.out.println("[SERVER] : recived a new
           table " + request);
504
505                     //      Processing table from client
506
507                     SERVER_GUI.txtout.append("[SERVER] : Processing
           table .... \n");
508
509                     String[] parts = request.split(",");
510                     int[] array = new int[parts.length];
511                     for (int i = 0; i < parts.length; i++) {
512                         array[i] = Integer.parseInt(parts[i]);
513                     }
514
515                     //      Sorting the array and finding repeated elements
516
517                     Arrays.sort(array);
518                     ArrayList<String> record = new ArrayList<String>
           >();
519
520                     for (int i = 0; i < array.length - 1; i++) {
521

```

```
522         if (array[i] == array[i + 1]) {
523             System.out.println("duplicate item [" +
524                 array[i + 1] + "]");
525             record.add(String.valueOf(array[i]));
526         }
527     }
528 }
529
530 //      Checking Results and sending them to client
531
532     if (record.isEmpty()) {
533         out.println("votre tableau ne contient
534             aucun doublon\n");
535         System.out.println("votre tableau ne
536             contient aucun doublon");
537         SERVER_GUI.txtout.append("[SERVER] :
538             Finished with NO Repeated elements \n");
539     } else {
540         SERVER_GUI.txtout.append("[SERVER] :
541             Finished. Found repeated " + record + "\n");
542         SERVER_GUI.txtout.append("[SERVER] :
543             Sending Results to Client\n");
544         out.println("votre tableau contient un
545             doublon\n duplicate items are : " +
546             record + "\n");
547         System.out.println("votre tableau ne
548             contient aucun doublon" + record);
549     }
550 }
551
552     }
553 }
554 } catch (IOException e) {
555     e.printStackTrace();
556 } finally {
557     out.close();
558     try {
559         in.close();
560     } catch (IOException e) {
561         e.printStackTrace();
562     }
563     try {
564         client.close();
565     } catch (IOException e) {
566         e.printStackTrace();
567     }
568 }
```

A.2 Java Code for CLIENT-GUI.java

```
569 package tp_01;
570
571
572 //TP1 Reseaux et Systemes Repartis 2021-2022
573
574 //Nom:HADJAZI
575 //Prenom: Mohammed Hisham
576 //Specialite:  RSSI      Groupe: 01
577
578 //Nom:Ameur
579 //Prenom: Wassim Malik
580 //Specialite:  RSSI      Groupe: 01
581
582
583 import java.awt.EventQueue;
584 import javax.swing.JFrame;
585 import javax.swing.JTextField;
586 import javax.swing.JLabel;
587 import javax.swing.SwingConstants;
588 import javax.swing.SwingWorker;
589 import javax.swing.JButton;
590 import java.awt.Font;
591 import java.awt.Color;
592 import java.awt.event.ActionListener;
593 import java.io.BufferedReader;
594 import java.io.IOException;
595 import java.io.InputStreamReader;
596 import java.io.PrintWriter;
597 import java.net.InetAddress;
598 import java.net.Socket;
599 import java.awt.event.ActionEvent;
600 import javax.swing.JTextArea;
601
602 public class CLIENT_GUI {
603
604     private JFrame frmClientTp;
605     private JTextField input1;
606     private JTextField input2;
607     private JTextField input3;
608     private PrintWriter out;
609     private BufferedReader in;
610     private Socket socket;
611
612     /**
613      * Launch the application.
614      */
615     public static void main(String[] args) {
616         EventQueue.invokeLater(new Runnable() {
617             public void run() {
618                 try {
619                     CLIENT_GUI window = new CLIENT_GUI();
620                     window.frmClientTp.setVisible(true);
621                 } catch (Exception e) {
622                     e.printStackTrace();
623                 }
624             }
625         });
626     }
627 }
```

```

624         }
625     });
626 }
627
628 /**
629  * Create the application.
630  */
631 public CLIENT_GUI() {
632     initialize();
633 }
634
635 /**
636  * Initialize the contents of the frame.
637  */
638 private void initialize() {
639     frmClientTp = new JFrame();
640     frmClientTp.getContentPane().setBackground(new Color(240,
641         255, 240));
642     frmClientTp.setBackground(new Color(102, 205, 170));
643     frmClientTp.setTitle("Client TP_01");
644     frmClientTp.setBounds(100, 100, 451, 491);
645     frmClientTp.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
646     frmClientTp.getContentPane().setLayout(null);
647
648     input1 = new JTextField();
649     input1.setFont(new Font("Dialog", Font.BOLD | Font.ITALIC,
650         26));
651     input1.setHorizontalAlignment(SwingConstants.CENTER);
652     input1.setBounds(215, 33, 208, 38);
653     frmClientTp.getContentPane().add(input1);
654     input1.setColumns(10);
655
656     input2 = new JTextField();
657     input2.setFont(new Font("Dialog", Font.BOLD | Font.ITALIC,
658         26));
659     input2.setHorizontalAlignment(SwingConstants.CENTER);
660     input2.setColumns(10);
661     input2.setBounds(215, 80, 208, 38);
662     frmClientTp.getContentPane().add(input2);
663
664     JLabel lblNewLabel = new JLabel("Enter server address :");
665     lblNewLabel.setHorizontalAlignment(SwingConstants.CENTER);
666     lblNewLabel.setBounds(35, 33, 159, 29);
667     frmClientTp.getContentPane().add(lblNewLabel);
668
669     JTextArea txtout = new JTextArea();
670     txtout.setForeground(Color.GREEN);
671     txtout.setBackground(Color.GRAY);
672     txtout.setBounds(35, 314, 366, 116);
673     frmClientTp.getContentPane().add(txtout);
674
675     JLabel lblEnterServerPort = new JLabel("Enter server port :
676         ");
677     lblEnterServerPort.setHorizontalAlignment(SwingConstants.
678         CENTER);
679     lblEnterServerPort.setBounds(35, 80, 159, 29);
680     frmClientTp.getContentPane().add(lblEnterServerPort);

```

```

676     JLabel label = new JLabel("");
677     label.setBounds(35, 130, 208, 40);
678     frmClientTp.getContentPane().add(label);
679
680
681     JButton btnStart = new JButton("Cennect");
682     btnStart.addActionListener(new ActionListener() {
683         public void actionPerformed(ActionEvent arg0) {
684
685             new SwingWorker() {
686
687                 @Override
688                 protected Object doInBackground() throws
689                     Exception {
690
691                     try {
692 //                     getting user input for server address and port
693
694                         int SERVER_PORT = Integer.parseInt(
695                             input2.getText());
696                         String SERVER_IP = input1.getText();
697 //                     Establishing socket
698
699                         socket = new Socket(SERVER_IP,
700                             SERVER_PORT);
701                         out = new PrintWriter(socket.
702                             getOutputStream(), true);
703                         in = new BufferedReader(new
704                             InputStreamReader(socket.
705                                 getInputStream()));
706
707 //                     getting Sever and client address and port
708
709                         InetAddress addrS = socket.
710                             getInetAddress();
711                         InetAddress addrC = socket.
712                             getLocalAddress();
713                         int portS = socket.getPort();
714                         int portC = socket.getLocalPort();
715
716                         txtout.setFont(new Font("Dialog", Font.
717                             BOLD, 9));
718                         txtout.setForeground(new Color(0, 255,
719                             0));
720                         txtout.append("[CLIENT] : My address is
721                             " + addrC + " port " + portC + "\n"
722                             );
723                         System.out.println("[CLIENT] : My
724                             address is " + addrC + " port " +
725                             portC);
726                         txtout.append("[CLIENT] : The server
727                             address is " + addrS + " port " +
728                             portS + "\n");

```

```

715         System.out.println("[CLIENT] : The
           server address is " + addrS + " port
           " + portS);
716
717         label.setFont(new Font("Dialog", Font.
           BOLD, 14));
718         label.setForeground(new Color(0, 255,
           0));
719         label.setText("Successful Connection");
720     } catch (Exception e) {
721         label.setFont(new Font("Dialog", Font.
           BOLD, 12));
722         label.setForeground(new Color(255, 0,
           0));
723         label.setText("ERROR" + e);
724     }
725
726     while (true) {
727
728         String serverResponse = in.readLine();
729         if (serverResponse.isEmpty()) {
730
731         } else {
732             System.out.println("[SERVER] : " +
               serverResponse);
733             txtout.setFont(new Font("Dialog",
               Font.BOLD, 9));
734             txtout.setForeground(new Color(0,
               255, 0));
735             txtout.append("[SERVER] : " +
               serverResponse + "\n");
736         }
737     }
738 }
739
740 }
741
742 }.execute();
743
744 }
745 });
746 btnStart.setBackground(new Color(175, 238, 238));
747 btnStart.setBounds(303, 128, 120, 29);
748 frmClientTp.getContentPane().add(btnStart);
749
750 JLabel lblResult = new JLabel("Result :");
751 lblResult.setHorizontalAlignment(SwingConstants.CENTER);
752 lblResult.setBounds(25, 273, 114, 29);
753 frmClientTp.getContentPane().add(lblResult);
754
755 input3 = new JTextField();
756 input3.setFont(new Font("Dialog", Font.BOLD, 20));
757 input3.setHorizontalAlignment(SwingConstants.CENTER);
758 input3.setBounds(35, 209, 388, 52);
759 frmClientTp.getContentPane().add(input3);
760 input3.setColumns(10);
761

```

```

762 JLabel lblEnterNumbers = new JLabel("Enter numbers : ex
    1,2,3,3,4");
763 lblEnterNumbers.setBounds(35, 182, 388, 15);
764 frmClientTp.getContentPane().add(lblEnterNumbers);
765
766 JButton btnSend = new JButton("Send");
767 btnSend.addActionListener(new ActionListener() {
768     public void actionPerformed(ActionEvent arg0) {
769
770         // getting input from user
771         String array = input3.getText();
772
773         // sending input to server
774         out.println(array);
775
776         // printing table
777         txtout.append("[CLIENT] : the array you provided is
            : " + array + "\n");
778
779         new SwingWorker() {
780
781             @Override
782             protected Object doInBackground() throws
                Exception {
783
784                 // Receiving solution from server
785
786                 try {
787
788                     while (true) {
789
790                         String serverResponse = in.readLine
791                             ();
792                         if (serverResponse.isEmpty()) {
793                             // empty response
794                         } else {
795                             System.out.println("[SERVER] :
                                " + serverResponse);
796                             txtout.setFont(new Font("Dialog
                                ", Font.BOLD, 9));
797                             txtout.setForeground(new Color
                                (0, 255, 0));
798                             txtout.append("[SERVER] : " +
                                serverResponse + "\n");
799                         }
800                     }
801
802                     } catch (Exception e) {
803                         e.printStackTrace();
804                     }
805
806                     return null;
807                 }
808
809             }.execute();
810

```

```
811         }
812     });
813     btnSend.setBackground(new Color(175, 238, 238));
814     btnSend.setBounds(303, 273, 120, 29);
815     frmClientTp.getContentPane().add(btnSend);
816
817     JButton btnStop = new JButton("Disconnect");
818     btnStop.addActionListener(new ActionListener() {
819         public void actionPerformed(ActionEvent arg0) {
820             out.println("bye");
821
822             try {
823                 socket.close();
824             } catch (IOException e) {
825             }
826
827             label.setFont(new Font("Dialog", Font.BOLD, 16));
828             label.setForeground(new Color(255, 0, 0));
829             label.setText("Disconnected");
830
831         }
832     });
833     btnStop.setBackground(new Color(175, 238, 238));
834     btnStop.setBounds(303, 168, 120, 29);
835     frmClientTp.getContentPane().add(btnStop);
836
837 }
838 }
```


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).