

Phần thực hành

Mục lục

I.	Hàm xử lý	2
1.	Kiểm tra số nguyên tố	2
2.	Kiểm tra số hoàn hảo	2
3.	Phân tích n thành thừa số nguyên tố.....	2
4.	Xóa những từ trùng nhau trong chuỗi	3
5.	Xử lý dữ liệu JSON Phần 1 (JSON trong Array)	3
6.	Xử lý dữ liệu JSON Phần 2 (Item trong JSON)	4
7.	Thực hiện phép tính $+ - * /$ bằng chuỗi.....	4
9.	Tra từ trong từ điển	5
10.	Đoán số	6
11.	Đảo ngược chuỗi	6
II.	TCP	6
1.	TCP Server	6
2.	TCP Client	7
III.	UDP	8
1.	UDP Server	8
2.	UDP Client	9
IV.	Multi-Thread	10
1.	Worker	10
2.	Server	10
3.	Client	11
V.	Chat với người lạ.....	12
1.	Interface	12
2.	Worker	12
3.	Server	15
4.	Client	16
VI.	Chat multiclients	18
1.	Worker	18
2.	Server	19
3.	Client	20

I. Hàm xử lý

1. Kiểm tra số nguyên tố

```
1. public static String kiemTraSoNguyenTo(int n) {
2.     if (n < 2) {
3.         return "Không phải là số nguyên tố.";
4.     } else {
5.         for (int i = 2; i <= Math.sqrt(n); i++) {
6.             if (n % i == 0) {
7.                 return "Không phải là số nguyên tố.";
8.             }
9.         }
10.    }
11.    return "Là số nguyên tố.";
12. }
```

2. Kiểm tra số hoàn hảo

```
1. private static String kiemTraSoHoanHao(int n) {
2.     for (int i = 1; (Math.pow(2, i) * (Math.pow(2, i + 1) - 1)) <= n; i++) {
3.         if ((Math.pow(2, i) * (Math.pow(2, i + 1) - 1)) == n) {
4.             return "Là số hoàn hảo.";
5.         }
6.     }
7.     return "Không phải là số hoàn hảo.";
8. }
```

3. Phân tích n thành thừa số nguyên tố

```
1. import java.util.*;
2.
3. public static boolean kiemTraSoNguyenTo(int n) {
4.     if (n < 2) {
5.         return false;
6.     } else {
7.         for (int i = 2; i <= Math.sqrt(n); i++) {
8.             if (n % i == 0) {
9.                 return false;
10.            }
11.        }
12.    }
13.    return true;
14. }
15.
16.
17. public static String phanTichThuaSoNguyenTo(int n) {
18.     HashMap < Integer, Integer > rs = new HashMap<Integer, Integer>();
19.     for (int i = 2; i <= n; i++) {
20.         while (kiemTraSoNguyenTo(i) && n % i == 0) {
21.             if (rs.get(i) != null) {
22.                 rs.put(i, (rs.get(i) + 1));
23.             } else {
24.                 rs.put(i, 1);
25.             }
26.             n /= i;
27.         }
28.     }
29.
30.     // Định dạng chuỗi
31.     String st = new String();
32.     for (int k : rs.keySet()) {
```

```

33.         st += k + "^" + rs.get(k) + " * ";
34.     }
35.     return st.substring(0, st.length() - 3);
36. }

```

4. Xóa những từ trùng nhau trong chuỗi

```

1.     import java.util.*;
2.
3.     public static String xoaTuLapLai(String input) {
4.         String output = "";
5.         LinkedHashMap < String, Integer > map = new LinkedHashMap<String, Integer>();
6.         StringTokenizer st = new StringTokenizer(data.toLowerCase());
7.         int i = 0;
8.         while (st.hasMoreTokens()) {
9.             String tmp = st.nextToken();
10.            map.put(tmp, ++i);
11.        }
12.        //Định dạng chuỗi trả về
13.        Iterator it = map.entrySet().iterator();
14.        while (it.hasNext()) {
15.            Map.Entry item = (Map.Entry)it.next();
16.            output += (it.hasNext()) ? item.getKey() + " _ " : item.getKey();
17.        }
18.        return output;
19.    }

```

5. Xử lý dữ liệu JSON Phần 1 (JSON trong Array)

```

//Dữ liệu trả về
{
  "coord": { ... }, // 2 items
  "weather": [
    {
      "id": 803,
      "main": "Clouds",
      "description": "broken clouds",
      "icon": "04d"
    }
  ],
  "base": "stations",
  "main": { ... }, // 6 items
  "visibility": 10000,
  "wind": { ... }, // 2 items
  "clouds": { ... }, // 1 item
  "dt": 1609407378,
  "sys": { ... }, // 5 items
  "timezone": 25200,
  "id": 1580578,
  "name": "Ho Chi Minh City",
  "cod": 200
}

// Lấy dữ liệu trong element weather:
1.     import java.io.IOException;
2.     import java.net.URL;
3.     import java.util.Scanner;
4.
5.     import org.json.*; //add json library
6.
7.     private static String getDataWeather() {
8.         try {
9.             URL url = new URL("https://api.openweathermap.org/data/2.5/weather?q=SaiGon&appid=8a40b8cd501f5deec9801dd7e8e5585d");
10.            Scanner sc = new Scanner(url.openStream());
11.            String inline = "";
12.            while (sc.hasNext())

```

```

13.         inline += sc.nextLine();
14.         sc.close();
15.
16.         JSONObject obj = new JSONObject(inline);
17.         JSONArray arr = (JSONArray) obj.get("weather");
18.         JSONObject objJSON = (JSONObject) arr.get(0);
19.         return objJSON.get("icon") + objJSON.get("id") + objJSON.get("main") + objJSON.get("description");
20.     } catch (IOException e) {
21.         return "Lỗi hệ thống.";
22.     }
23. }

```

6. Xử lý dữ liệu JSON Phần 2 (Item trong JSON)

```

// Dữ liệu trả về
{
  "coord": {
    "lon": 106.67,
    "lat": 10.83
  },
  "weather": [ ... ], // 1 item
  "base": "stations",
  "main": { ... }, // 6 items
  "visibility": 10000,
  "wind": { ... }, // 2 items
  "clouds": { ... }, // 1 item
  "dt": 1609466851,
  "sys": { ... }, // 5 items
  "timezone": 25200,
  "id": 1580578,
  "name": "Ho Chi Minh City",
  "cod": 200
}

// Ví dụ lấy dữ liệu trong element coord
1.     import java.io.IOException;
2.     import java.net.URL;
3.     import java.util.Scanner;
4.
5.     import org.json.*; //add json library
6.
7.     private static String getDataCoord() {
8.         try {
9.             URL url = new URL("https://api.openweathermap.org/data/2.5/weather?q=SaiGon&appid=8a40b8cd501f5deec9801dd7e8e5585d");
10.            Scanner sc = new Scanner(url.openStream());
11.            String inline = "";
12.            while (sc.hasNext())
13.                inline += sc.nextLine();
14.            sc.close();
15.
16.            JSONObject obj = new JSONObject(inline);
17.            JSONObject objJSON = (JSONObject) obj.get("coord");
18.            return objJSON.get("lon") + " " + objJSON.get("lat");
19.        } catch (IOException e) {
20.            return "Lỗi hệ thống.";
21.        }
22.    }

```

7. Thực hiện phép tính +-* / bằng chuỗi

```

1.     import java.util.*;
2.
3.     public static String Cau1(String input) {
4.         try {
5.             StringTokenizer st = new StringTokenizer(input, "+-*/", true);
6.             float rs = Float.parseFloat(st.nextToken());
7.             while (st.hasMoreTokens()) {
8.                 String next = st.nextToken();
9.                 if (next.equals("+")) {

```

```

10.         rs += Float.parseFloat(st.nextToken());
11.     } else if (next.equals("-")) {
12.         rs -= Float.parseFloat(st.nextToken());
13.     } else if (next.equals("*")) {
14.         rs *= Float.parseFloat(st.nextToken());
15.     } else if (next.equals("/")) {
16.         rs /= Float.parseFloat(st.nextToken());
17.     } else {
18.         return "Định dạng không đúng. Cần kiểm tra lại.";
19.     }
20. }
21. return rs + "";
22. } catch (NumberFormatException nfe) {
23.     return "Định dạng không đúng. Cần kiểm tra lại.";
24. }
25. }

```

8. Đọc JSON từ file

```

// Cấu trúc file
file.json - Notepad
File Edit Format View Help
{
  "Name" : "Thanh Hai",
  "Course": "Lap trình mạng",
  "Subject": [
    "mạng máy tính",
    "lập trình mạng",
    "mà nguồn mō"
  ]
}

// Xử lý
1. import java.io.FileReader;
2. import org.json.simple.*;
3. import org.json.simple.parser.*;
4.
5. private static String getDataFile() {
6.     JSONParser parser = new JSONParser();
7.     try {
8.         Object obj = parser.parse(new FileReader("file.json"));
9.         JSONObject jsonObject = (JSONObject)obj;
10.        //Lấy value của name
11.        String name = (String)jsonObject.get("Name");
12.        //Lấy value của course
13.        String course = (String)jsonObject.get("Course");
14.        //Lấy value của subject
15.        JSONArray subject = (JSONArray)jsonObject.get("Subject");
16.
17.        return name + " " + course + " " + subject;
18.    } catch (Exception e) {
19.        return "Lỗi hệ thống.";
20.    }
21. }

```

9. Tra từ trong từ điển

```

1. import java.io.File;
2. import java.util.*;
3.
4. private String traTu(String input) {
5.     try {
6.         File myObj = new File("./src/txt.txt");
7.         Scanner scanner = new Scanner(myObj);
8.
9.         HashMap < String, String > map1 = new HashMap<String, String>();
10.        HashMap < String, String > map2 = new HashMap<String, String>();

```

```

11.
12.     while (scanner.hasNextLine()) {
13.         String data = scanner.nextLine();
14.         String[] tmp = data.split(",");
15.         map1.put(tmp[0], tmp[1]);
16.         map2.put(tmp[1], tmp[0]);
17.     }
18.     scanner.close();
19.
20.     for (Map.Entry < String, String > entry : map1.entrySet()) {
21.         if (entry.getKey().equalsIgnoreCase(input)) {
22.             return entry.getValue();
23.         }
24.     }
25.     for (Map.Entry < String, String > entry : map2.entrySet()) {
26.         if (entry.getKey().equalsIgnoreCase(input)) {
27.             return entry.getValue();
28.         }
29.     }
30. } catch (Exception e) {
31.     return "Lỗi hệ thống.";
32. }
33. return "Không tìm thấy từ trong từ điển 😞😞";
34. }

```

10. Đoán số

```

1.     // Sinh số ngẫu nhiên từ 1 đến <= 100
2.     private static int soNgauNhan = (int)(Math.random() * 100 + 1);
3.
4.     private static String doanSo(String input) {
5.         try {
6.             int so = Integer.parseInt(input);
7.             System.out.println("Số ngẫu nhiên là " + soNgauNhan);
8.             if (so == soNgauNhan) {
9.                 return "Chính xác.";
10.            } else if (so < soNgauNhan) {
11.                return "Cần gửi số lớn hơn.";
12.            } else {
13.                return "Cần gửi số bé hơn.";
14.            }
15.        } catch (NumberFormatException nfe) {
16.            return "Không phải là số.";
17.        }
18.    }

```

11. Đảo ngược chuỗi

```

1.     private static String daoNguocChuoi(String input) {
2.         String rs = "";
3.         String[] tmp = input.split(" ");
4.         for (int i = 0; i < tmp.length; i++) {
5.             rs += new StringBuilder(tmp[i]).reverse().toString();
6.             if (i == tmp.length - 1) continue;
7.             rs += " ";
8.         }
9.         return rs;
10.    }

```

II. TCP

1. TCP Server

```

1. import java.io.*;
2. import java.net.*;
3.
4. public class Server {
5.
6.     //Viết hàm Xử lý logic tại đây (Lưu ý: Trả về String)
7.     private String xuly(String input) {
8.         return "";
9.     }
10.
11.     private Server(int port) {
12.         try {
13.             ServerSocket server = new ServerSocket(port);
14.             System.out.println("Server started");
15.             System.out.println("Waiting for a client...");
16.             Socket socket = server.accept();
17.             System.out.println("Client accepted");
18.
19.             BufferedWriter out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
20.             BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
21.
22.             String line = ""; // Nội dung Server nhận từ Client
23.             String dataSend = ""; // Nội dung Server gửi về client
24.
25.             do {
26.                 line = in.readLine(); //Nhận data từ client gửi
27.                 dataSend = xuly(line); // Data gửi về client là nội dung hàm xử lý
28.
29.                 //Server gửi data về client
30.                 out.write(dataSend);
31.                 out.newLine();
32.                 out.flush();
33.             } while (!line.equalsIgnoreCase("bye"));
34.
35.             //CLOSE
36.             in.close();
37.             out.close();
38.             socket.close();
39.             server.close();
40.             System.err.println("Server closed");
41.         } catch (IOException e) {
42.             e.printStackTrace();
43.         }
44.     }
45.
46.     public static void main(String[] args) {
47.         Server server = new Server(5000);
48.     }
49. }

```

2. TCP Client

```

1. import java.io.*;
2. import java.net.Socket;
3.
4. public class Client {
5.     private Client(String address, int port) {
6.         try {
7.             Socket socket = new Socket(address, port);
8.             System.out.println("Connect to server successful");
9.
10.             BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
11.             BufferedWriter out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
12.             BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));

```

```

13.
14.     String line = ""; // Nội dung Client gửi lên Server
15.
16.     do {
17.         // Nhập data gửi lên Server
18.         System.out.print("Enter content: ");
19.         line = stdIn.readLine();
20.         if (line.equalsIgnoreCase("bye")) {
21.             break;
22.         }
23.
24.         // Client gửi data lên Server
25.         out.write(line);
26.         out.newLine();
27.         out.flush();
28.
29.         // Nhận data từ Server gửi về
30.         String dt = in.readLine();
31.         if (dt != null) {
32.             System.out.println("Ket qua: " + dt);
33.         }
34.     } while (!line.equalsIgnoreCase("bye"));
35.
36.     //CLOSE
37.     System.out.print("See you later ❤️❤️");
38.     in.close();
39.     out.close();
40.     socket.close();
41. } catch (IOException e) {
42.     e.printStackTrace();
43. }
44. }
45.
46. public static void main(String[] args) {
47.     Client client1 = new Client("127.0.0.1", 5000);
48. }
49. }

```

III. UDP

1. UDP Server

```

1.     import java.io.IOException;
2.     import java.net.*;
3.
4.     public class Server {
5.         private static int buffsize = 512;
6.         private static int port = 1234;
7.
8.         private static String xuly(String input) {
9.             return "";
10.        }
11.
12.        public static void main(String[] args) {
13.            DatagramSocket socket;
14.            DatagramPacket dpreceive, dpsend;
15.            try {
16.                socket = new DatagramSocket(port);
17.                dpreceive = new DatagramPacket(new byte[buffsize], buffsize);
18.                System.out.println("Server is ready!");
19.                while (true) {
20.                    // Server nhận dữ liệu từ Client
21.                    socket.receive(dpreceive);
22.                    String dataReceived = new String(dpreceive.getData(), 0, dpreceive.getLength());

```



```

23.         System.out.println("Server received" + dataReceived);
24.
25.         //catch keyword stop server
26.         if (dataReceived.equals("bye")) {
27.             System.out.println("Server closed!");
28.             socket.close();
29.             break;
30.         }
31.
32.         String dataSend = xuly(dataReceived); // Nội dung Server gửi về Client
33.
34.         // Server gửi data về Client
35.         dpsend = new DatagramPacket(dataSend.getBytes(), dataSend.getBytes().length, dpreceive.getAddress(), dpreceive.getPort());
36.         System.out.println("Server sent: " + dataSend);
37.         socket.send(dpsend);
38.     }
39. } catch (IOException e) {
40.     System.err.println(e);
41. }
42. }
43. }

```

2. UDP Client

```

1. import java.io.IOException;
2. import java.net.DatagramPacket;
3. import java.net.DatagramSocket;
4. import java.net.InetAddress;
5. import java.util.Scanner;
6.
7. public class Client {
8.
9.     private static int destPort = 1234;
10.    private static String hostname = "localhost";
11.
12.    public static void main(String[] args) {
13.        DatagramSocket socket;
14.        DatagramPacket dpsend, dpreceive;
15.        InetAddress add; Scanner stdIn;
16.        System.out.println("Client is ready!");
17.
18.        try {
19.            add = InetAddress.getByName(hostname);
20.            socket = new DatagramSocket();
21.            stdIn = new Scanner(System.in);
22.
23.            while (true) {
24.                // Nhập nội dung gửi tới Server
25.                System.out.print("Client input: ");
26.                String dataSend = stdIn.nextLine();
27.                byte[] data = dataSend.getBytes();
28.
29.                // Client gửi data tới Server
30.                dpsend = new DatagramPacket(data, data.length, add, destPort);
31.                System.out.println("Client sent: " + dataSend);
32.                socket.send(dpsend);
33.
34.                //catch stop server
35.                if (dataSend.equals("bye")) {
36.                    System.out.println("Client socket closed!");
37.                    stdIn.close();
38.                    socket.close();
39.                    break;
40.                }
41.            }

```

```

42.         // Client nhận data từ Server
43.         dpreceive = new DatagramPacket(new byte[512], 512);
44.         socket.receive(dpreceive);
45.         String dataReceived = new String(dpreceive.getData(), 0, dpreceive.getLength());
46.         System.out.println("Client get: " + dataReceived);
47.     }
48. } catch (IOException e) { System.err.println(e); }
49. }
50. }

```

IV. Multi-Thread

1. Worker

```

1.  import java.io.*;
2.  import java.net.Socket;
3.
4.  public class Worker implements Runnable {
5.      private Socket socket;
6.      BufferedReader in;
7.      BufferedWriter out;
8.
9.      private static String xuly(String input) {
10.         return "Kết quả nè: ";
11.     }
12.
13.     public Worker(Socket s) throws IOException {
14.         this.socket = s;
15.         this.in = new BufferedReader(new InputStreamReader(s.getInputStream()));
16.         this.out = new BufferedWriter(new OutputStreamWriter(s.getOutputStream()));
17.     }
18.
19.     public void run() {
20.         System.out.println("Client " + socket.toString() + " accepted");
21.         try {
22.             String input = "";
23.             while (true) {
24.                 //Server nhận data từ Client
25.                 input = in.readLine();
26.                 System.out.println("Server received: " + input);
27.                 if (input.equals("bye")) {
28.                     break;
29.                 }
30.                 // Server gửi data về Client
31.                 String output = xuly(input);
32.                 out.write(output + '\n');
33.                 out.flush();
34.                 System.out.println("Server write: " + input);
35.             }
36.             System.out.println("Closed socket " + socket.toString());
37.             in.close();
38.             out.close();
39.             socket.close();
40.         } catch (IOException e) {
41.             System.out.println(e);
42.         }
43.     }
44. }

```

2. Server

```

1.  import java.io.IOException;
2.  import java.net.*;
3.  import java.util.Vector;

```

```

4.     import java.util.concurrent.*;
5.
6.     public class Server {
7.         public static int port = 1234;
8.         public static int numThread = 2;
9.         private static ServerSocket server = null;
10.        public static Vector<Worker> workers = new Vector<>();
11.
12.        public static void main(String[] args) throws IOException {
13.            ExecutorService executor = Executors.newFixedThreadPool(numThread);
14.            try {
15.                server = new ServerSocket(port);
16.                System.out.println("Server binding at port " + port);
17.                System.out.println("Waiting for client...");
18.                while (true) {
19.                    Socket socket = server.accept();
20.                    Worker client = new Worker(socket);
21.                    executor.execute(client);
22.                }
23.            } catch (IOException e) {
24.                System.out.println(e);
25.            } finally {
26.                if (server != null)
27.                    server.close();
28.            }
29.        }
30.    }

```

3. Client

```

1.     import java.io.*;
2.     import java.net.Socket;
3.     import java.util.concurrent.ExecutorService;
4.     import java.util.concurrent.Executors;
5.
6.     class SendMessage implements Runnable {
7.         private BufferedWriter out;
8.         private Socket socket;
9.         public SendMessage(Socket s, BufferedWriter o) {
10.            this.socket = s;
11.            this.out = o;
12.        }
13.        public void run() {
14.            try {
15.                while (true) {
16.
17.                    BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
18.                    String data = stdIn.readLine();
19.                    System.out.println("Input from client: " + data);
20.                    out.write(data + '\n');
21.                    out.flush();
22.                    if (data.equals("bye"))
23.                        break;
24.                }
25.                System.out.println("Client closed connection");
26.                out.close();
27.                socket.close();
28.            } catch (IOException e) { }
29.        }
30.    }
31.
32.    class ReceiveMessage implements Runnable {
33.        private BufferedReader in;
34.        private Socket socket;
35.        public ReceiveMessage(Socket s, BufferedReader i) {

```

```

36.         this.socket = s;
37.         this.in = i;
38.     }
39.     public void run() {
40.         try {
41.             while (true) {
42.                 String data = in.readLine();
43.                 System.out.println("Receive: " + data);
44.             }
45.         } catch (IOException e) {}
46.     }
47. }
48.
49. public class Client {
50.     private static String host = "localhost";
51.     private static int port = 1234;
52.     private static Socket socket;
53.
54.     private static BufferedWriter out;
55.     private static BufferedReader in;
56.
57.     public static void main(String[] args) throws IOException {
58.         socket = new Socket(host, port);
59.         System.out.println("Client connected");
60.         out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
61.         in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
62.         ExecutorService executor = Executors.newFixedThreadPool(2);
63.         SendMessage send = new SendMessage(socket, out);
64.         ReceiveMessage recv = new ReceiveMessage(socket, in);
65.         executor.execute(send);
66.         executor.execute(recv);
67.     }
68. }

```

V. Chat với người lạ

1. Interface

```

1.     public interface Interface {
2.     }
3.
4.     interface IListenerDisconnect{
5.         void onDisconnect();
6.     }
7.     interface IListenerLogin{
8.         void onLoginSuccess();
9.         void onLoginFailed();
10.        void onConnectedAnotherClient();
11.        void onDisconnectedAnotherClient();
12.    }

```

2. Worker

```

1.     import java.io.*;
2.     import java.net.Socket;
3.
4.     public class Worker implements Runnable {
5.         private String myName;
6.         private Socket socket;
7.         BufferedReader in;
8.         BufferedWriter out;
9.         private String stateConnect = "notLogin";//connected or waiting
10.        private String connectedUserName;

```

```

11.
12.     public Worker(Socket s, String name) throws Exception {
13.         this.socket = s;
14.         this.myName = name;
15.         this.in = new BufferedReader(new InputStreamReader(s.getInputStream()));
16.         this.out = new BufferedWriter(new OutputStreamWriter(s.getOutputStream()));
17.         this.connectedUserName = "";
18.     }
19.     //TODO: Failed login but connect to client
20.     public void run() {
21.         System.out.println("Client " + socket.toString() + " accepted");
22.         try {
23.             while (true) {
24.                 String[] arrInput = null;
25.                 String data = receiveDataFromClient();
26.                 if (data.equals("bye")) {
27.                     send("bye");
28.                     if (stateConnect.equals("connected")) sendUniCast(connectedUserName, "disconnect");
29.                     break;
30.                 }
31.                 if (data.equals("disconnected"))
32.                 {
33.                     stateConnect = "waiting";
34.                     send("Connected client disconnect!");
35.                 }
36.                 arrInput = data.split("#");//Validate input
37.                 if (arrInput.length < 2 && stateConnect.equals("notLogin")) {
38.                     this.send("Du lieu khong hop le.");
39.                     continue;
40.                 }
41.
42.                 switch (stateConnect) {
43.                     case "notLogin": {
44.                         handleNotLogin(arrInput);
45.                     }
46.                     break;
47.                     case "waiting": {
48.                         this.send("Waiting for another client ....");
49.                         handleWaiting();
50.                         //connect to waiting client
51.                     }
52.                     break;
53.                     case "connected": {
54.                         handleConnected(data);
55.                         //Send msg to connected client
56.                     }
57.                     break;
58.                 }
59.             }
60.             //Client close connection
61.             System.out.println("Closed socket for client " + myName + " " + socket.toString());
62.             in.close();
63.             out.close();
64.             socket.close();
65.             Server.workers.remove(this);
66.         } catch (Exception e) {
67.             e.printStackTrace();
68.         }
69.     }
70.
71.
72.     private void handleConnected(String data) {
73.         this.sendUniCast(this.connectedUserName, data);
74.     }
75.
76.     public void handleNotLogin(String[] arrInput) {

```

```

77.     String optionalMessage = arrInput[0];
78.     String messageClient = arrInput[1];
79.     switch (optionalMessage) {
80.         case "all": {
81.             if (sendBroadCast(messageClient)) {
82.                 this.send("Send message success!");
83.             } else {
84.                 this.send("Send message error!");
85.             }
86.         }
87.         break;
88.         case "login": {
89.             if (validateLogin(messageClient)) {
90.                 this.myName = messageClient;
91.                 this.send("SuccessLogin");
92.                 this.stateConnect = "waiting";
93.             } else {
94.                 this.send("Failed Login");
95.                 break;
96.             }
97.             // connect success
98.             if (connectToAnotherClient()) {
99.                 this.send("Connected to " + connectedUserName);
100.            } else {
101.                this.send("Waiting for another client .....");
102.            }
103.        }
104.        break;
105.        default: {
106.            // Default send msg to another user
107.            if (sendUniCast(optionalMessage, messageClient)) {
108.                this.send("Send message success!");
109.            } else {
110.                this.send("Send message error");
111.            }
112.        }
113.    }
114. }
115.
116. public void handleWaiting() {
117.     while (true) {
118.         // connect success
119.         if (connectToAnotherClient()) {
120.             this.send("Connected to " + connectedUserName);
121.             break;
122.         }
123.     }
124. }
125.
126. public boolean sendBroadCast(String input) {
127.     for (Worker worker : Server.workers) {
128.         if (!myName.equals(worker.myName)) {
129.             if (!worker.send(input))
130.                 return false;
131.         }
132.     }
133.     return true;
134. }
135.
136. public String receiveDataFromClient() {
137.     String input = "";
138.     try {
139.         input = in.readLine();
140.         System.out.println("Server received: " + input + " from " + socket.toString() + " # Client " + myName);
141.         return input;
142.     } catch (Exception e) {

```

```

143.         return "Exception\n";
144.     }
145. }
146.
147. public boolean sendUniCast(String nameWorker, String input) {
148.     for (Worker worker : Server.workers) {
149.         if (nameWorker.equals(worker.myName)) {
150.             if (worker.send(input)) return true;
151.             else break;
152.         }
153.     }
154.     return false;
155. }
156.
157. public boolean validateLogin(String userName) {
158.     for (Worker worker : Server.workers) {
159.         if (userName.equals(worker.myName))
160.             return false; // userName exists
161.     }
162.     return true;
163. }
164.
165. public boolean send(String data) {
166.     try {
167.         this.out.write(data + '\n');
168.         this.out.flush();
169.         return true;
170.     } catch (Exception e) {
171.         System.out.println(e + "ERROR Send msg to client ");
172.         return false;
173.     }
174. }
175.
176. public boolean connectToAnotherClient() {
177.     for (Worker worker : Server.workers) {
178.         if (worker.stateConnect.equals("waiting")) {
179.             // two same worker not connect
180.             if (!worker.myName.equals(this.myName)) {
181.                 //add connectedUserName
182.                 worker.connectedUserName = this.myName;
183.                 this.connectedUserName = worker.myName;
184.                 //Change state
185.                 worker.stateConnect = "connected";
186.                 this.stateConnect = "connected";
187.                 //send announcement to client of worker connected
188.                 worker.send("Connected to " + worker.connectedUserName);
189.                 worker.send("Please enter message to send " + worker.connectedUserName);
190.                 return true;
191.             }
192.         }
193.     }
194.     return false;
195. }
196. }

```

3. Server

```

1.     import java.net.ServerSocket;
2.     import java.net.Socket;
3.     import java.util.Vector;
4.     import java.util.concurrent.ExecutorService;
5.     import java.util.concurrent.Executors;
6.
7.     public class Server {
8.         public static int port = 1234;

```

```

9.     public static int numThread = 4;
10.    private static ServerSocket server = null;
11.    public static Vector<Worker> workers = new Vector<>();
12.
13.    public static void main(String[] args) throws Exception {
14.        int i = 0;
15.        ExecutorService executor = Executors.newFixedThreadPool(numThread);
16.        try {
17.            server = new ServerSocket(port);
18.            System.out.println("Server binding at port " + port);
19.            System.out.println("Waiting for client...");
20.            while(true) {
21.                i++;
22.                Socket socket = server.accept();
23.                Worker client = new Worker(socket, Integer.toString(i));
24.                workers.add(client);
25.                executor.execute(client);
26.            }
27.        } catch (Exception e) {
28.            System.out.println(e);
29.        } finally {
30.            if(server!=null)
31.                server.close();
32.        }
33.    }
34. }

```

4. Client

```

1.    import java.io.*;
2.    import java.net.Socket;
3.    import java.util.concurrent.ExecutorService;
4.    import java.util.concurrent.Executors;
5.
6.    class SendMessage implements Runnable {
7.        private BufferedWriter out;
8.        private Socket socket;
9.        public boolean isLogin = false;
10.        public String msgWorker = "";
11.        public SendMessage(Socket s, BufferedWriter o) {
12.            this.socket = s;
13.            this.out = o;
14.        }
15.        public void run() {
16.            System.out.println("Please enter your username: ");
17.            try {
18.                BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
19.                while(true) {
20.                    String data = "";
21.                    data += stdIn.readLine();
22.                    if(msgWorker.equals("disconnected")) {
23.                        data = "disconnected";
24.                        msgWorker = "";
25.                    }
26.                    if(!isLogin){
27.                        System.out.println("Please enter your username: ");
28.                        String temp = data;
29.                        data = "login#" + temp;
30.                        System.out.println("Input from client: " + data);
31.                    }else {
32.                        System.out.println("Your message: " + data);
33.                    }
34.
35.                    out.write(data + '\n');
36.                    out.flush();

```



```

37.         if(data.equals("bye")) {
38.             stdIn.close();
39.             break;
40.         }
41.     }
42.     out.close();
43. } catch (Exception e) {}
44. }
45. }
46.
47. class ReceiveMessage implements Runnable {
48.     private BufferedReader in;
49.     private Socket socket;
50.     private IListenerDisconnect iListenerDisconnect;
51.     private IListenerLogin iListenerLogin;
52.     public ReceiveMessage(Socket s, BufferedReader i, IListenerDisconnect iListenerDisconnect, IListenerLogin iListenerLogin) {
53.         this.socket = s;
54.         this.in = i;
55.         this.iListenerDisconnect = iListenerDisconnect;
56.         this.iListenerLogin = iListenerLogin;
57.     }
58.     public void run() {
59.         try {
60.             while(true) {
61.                 String data = in.readLine();
62.                 System.out.println("Receive: " + data);
63.                 if(data.equals("bye")) {
64.                     System.out.println("Exit program");
65.                     iListenerDisconnect.onDisconnect();
66.                     break;
67.                 }
68.                 if(data.equals("SuccessLogin")) iListenerLogin.onLoginSuccess();
69.                 if(data.equals("disconnect")) iListenerLogin.onDisconnectedAnotherClient();
70.             }
71.             System.out.println("Close connection");
72.             in.close();
73.             socket.close();
74.         } catch (Exception e) {}
75.     }
76. }
77. }
78.
79. public class Client {
80.     private static String host = "localhost";
81.     private static int port = 1234;
82.     private static Socket socket;
83.     private static BufferedWriter out;
84.     private static BufferedReader in;
85.
86.     public static void main(String[] args) throws Exception {
87.         socket = new Socket(host, port);
88.         System.out.println("Client connected");
89.         out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
90.         in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
91.         ExecutorService executor = Executors.newFixedThreadPool(2);
92.         SendMessage send = new SendMessage(socket, out);
93.
94.         ReceiveMessage recv = new ReceiveMessage(socket, in, () -> executor.shutdownNow(), new IListenerLogin() {
95.             @Override
96.             public void onLoginSuccess() {
97.                 send.isLogin = true;
98.             }
99.
100.             @Override
101.             public void onLoginFailed() {
102.

```

```

103.         }
104.
105.         @Override
106.         public void onConnectedAnotherClient() {
107.         }
108.
109.         @Override
110.         public void onDisconnectedAnotherClient() {
111.             send.msgWorker = "disconnected";
112.         }
113.     });
114.     executor.execute(send);
115.     executor.execute(rcv);
116.
117. }
118. }

```

VI. Chat multiciens

1. Worker

```

1.  import java.io.*;
2.  import java.net.Socket;
3.  import java.util.regex.Pattern;
4.
5.  public class Worker implements Runnable {
6.      private String myName;
7.      private Socket socket;
8.      BufferedReader in;
9.      BufferedWriter out;
10.
11.     public Worker(Socket s, String name) throws IOException {
12.         this.socket = s;
13.         this.myName = name;
14.         this.in = new BufferedReader(new InputStreamReader(s.getInputStream()));
15.         this.out = new BufferedWriter(new OutputStreamWriter(s.getOutputStream()));
16.     }
17.
18.     public void run() {
19.         System.out.println("Client " + socket.toString() + " is accepted");
20.         try {
21.             sendUnicast(myName, "NAME#" + myName);
22.             sendBroadcast(myName + " is online");
23.             String input = "";
24.             while (true) {
25.                 input = in.readLine();
26.                 System.out.println("Server received: " + input + " from " + socket.toString() + " # Client " + myName);
27.                 if (input.equals("bye")) {
28.                     break;
29.                 }
30.                 else if (input.startsWith("all#")) {
31.                     String msg = input.substring(4);
32.                     out.write(sendBroadcast(msg));
33.                     out.flush();
34.                 } else if (checkInput(input.split("#")[0])) {
35.                     String[] data = input.split("#");
36.                     out.write(sendUnicast(data[0], input.substring(data[0].length() + 1)));
37.                     out.flush();
38.                 } else {
39.                     out.write("Sai định dạng");
40.                 }
41.             }
42.             System.out.println("Closed socket for client " + myName + " " + socket.toString());
43.             sendBroadcast(myName + " is offline");
44.             in.close();

```

```

45.         out.close();
46.         socket.close();
47.         removeWorker(myName);
48.     } catch (IOException e) {
49.         System.out.println(e);
50.     }
51. }
52.
53. private boolean checkInput(String input) {
54.     Pattern pattern = Pattern.compile("^\\d+");
55.     return pattern.matcher(input).matches();
56. }
57.
58. private String sendBroadcast(String msg){
59.     try {
60.         for (Worker worker : Server.workers) {
61.             if (!myName.equals(worker.myName)) {
62.                 worker.out.write(msg + '\n');
63.                 worker.out.flush();
64.             }
65.         }
66.     } catch (Exception e) {
67.         return "Can't send broadcast\n";
68.     }
69.     return "Sent broadcast successful\n";
70. }
71.
72. private String sendUnicast(String name, String msg) throws IOException {
73.     try {
74.         for (Worker worker : Server.workers) {
75.             if (name.equals(worker.myName)) {
76.                 worker.out.write(msg + '\n');
77.                 worker.out.flush();
78.                 return "Sent message successful\n"; // success
79.             }
80.         }
81.     } catch (Exception e) {
82.         return "Exception\n";
83.     } // exception
84.     return "Can't send broadcast\n"; // client not found
85. }
86.
87. private boolean removeWorker(String name) {
88.     try {
89.         for (Worker worker : Server.workers)
90.             if (name.equals(worker.myName)) {
91.                 Server.workers.remove(worker);
92.                 break;
93.             }
94.     } catch (Exception e) {
95.         return false;
96.     }
97.     return true;
98. }
99. }

```

2. Server

```

1. import java.io.IOException;
2. import java.net.ServerSocket;
3. import java.net.Socket;
4. import java.util.Vector;
5. import java.util.concurrent.ExecutorService;
6. import java.util.concurrent.Executors;
7.

```

```

8.     public class Server {
9.         public static int port = 1234;
10.        public static int numThread = 10;
11.        public static ServerSocket server = null;
12.        public static Vector<Worker> workers = new Vector<>();
13.
14.        public static void main(String[] args) throws IOException {
15.            int i = 0;
16.            ExecutorService executor = Executors.newFixedThreadPool(numThread);
17.            try {
18.                server = new ServerSocket(port);
19.                System.out.println("Server binding at port " + port);
20.                System.out.println("Waiting for clients...");
21.                while (true) {
22.                    i++;
23.                    Socket socket = server.accept();
24.                    Worker client = new Worker(socket, Integer.toString(i));
25.                    workers.add(client);
26.                    executor.execute(client);
27.                }
28.            } catch (IOException e) {
29.                System.out.println(e);
30.            } finally {
31.                if (server != null)
32.                    server.close();
33.            }
34.        }
35.    }

```

3. Client

```

1.     import java.io.*;
2.     import java.net.Socket;
3.     import java.util.concurrent.ExecutorService;
4.     import java.util.concurrent.Executors;
5.
6.     class SendMessage implements Runnable {
7.         private BufferedWriter out;
8.         private Socket socket;
9.         public SendMessage(Socket s, BufferedWriter o) {
10.            this.socket = s;
11.            this.out = o;
12.        }
13.        public void run() {
14.            try {
15.                while (true) {
16.                    BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
17.                    String data = stdIn.readLine();
18.                    out.write(data + '\n');
19.                    out.flush();
20.                    if (data.equals("bye"))
21.                        break;
22.                }
23.                System.out.println("CLIENT " + Client.myName + " closed connection");
24.                out.close();
25.                socket.close();
26.                Client.executor.shutdownNow();
27.            } catch (IOException e) { }
28.        }
29.    }
30.
31.    class ReceiveMessage implements Runnable {
32.        private BufferedReader in;
33.        private Socket socket;
34.        public ReceiveMessage(Socket s, BufferedReader i) {

```

```

35.         this.socket = s;
36.         this.in = i;
37.     }
38.     public void run() {
39.         try {
40.             while (true) {
41.                 String data = in.readLine();
42.                 if (data.startsWith("NAME#")) {
43.                     Client.myName = data.split("#")[1];
44.                     System.out.println("CLIENT " + Client.myName + " connected");
45.                     continue;
46.                 }
47.                 if (data.equals("bye"))
48.                     break;
49.                 System.out.println("CLIENT " + Client.myName + " received: " + data);
50.             }
51.             socket.close();
52.         } catch (IOException e) { }
53.     }
54. }
55.
56. public class Client {
57.     public static ExecutorService executor;
58.     private static String host = "localhost";
59.     private static int port = 1234;
60.     private static Socket socket;
61.     public static String myName = "";
62.
63.     private static BufferedWriter out;
64.     private static BufferedReader in;
65.
66.     public static void main(String[] args) throws IOException, InterruptedException {
67.         socket = new Socket(host, port);
68.         out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
69.         in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
70.         executor = Executors.newFixedThreadPool(2);
71.         SendMessage send = new SendMessage(socket, out);
72.         ReceiveMessage recv = new ReceiveMessage(socket, in);
73.         executor.execute(send);
74.         executor.execute(recv);
75.     }
76. }

```