Phần thực hành

Mục lục

I. Hàm xử lý

1. Kiểm tra số nguyên tố

```
public static String kiemTraSoNguyenTo(int n) {
  if (n < 2) {
    return "Không phải là số nguyên tố.";
  } else {
    for (int i = 2; i <= Math.sqrt(n); i++) {
      if (n % i == 0) {
        return "Không phải là số nguyên tố.";
      }
    }
  }
  return "Là số nguyên tố.";
}
```

2. Kiểm tra số hoàn hảo

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

3. Phân tích n thành thừa số nguyên tố

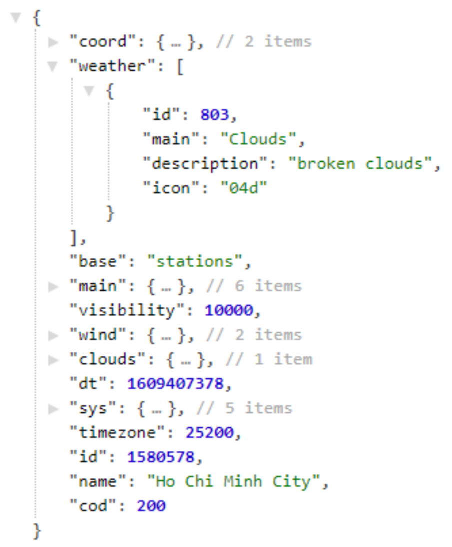
```
import java.util.*;

public static boolean kiemTraSoNguyenTo(int n) {
  if (n < 2) {
    return false;
  } else {
    for (int i = 2; i <= Math.sqrt(n); i++) {
      if (n % i == 0) {
        return false;
      }
    }
  }
  return true;
}


public static String phanTichThuaSoNguyenTo(int n) {
  HashMap < Integer, Integer > rs = new HashMap<Integer, Integer>();
  for (int i = 2; i <= n ; i++) {
    while (kiemTraSoNguyenTo(i) && n % i == 0) {
      if (rs.get(i) != null) {
        rs.put(i, (rs.get(i) + 1));
      } else {
        rs.put(i, 1);
      }
      n /= i;
    }
  }

  // Định dạng chuỗi
  String st = new String();
  for (int k : rs.keySet()) {
    st += k + "^" + rs.get(k) + " * ";
  }
  return st.substring(0, st.length() - 3);
}
```

4. Xóa những từ trùng nhau trong chuỗi

```
import java.util.*;

public static String xoaTuLapLai(String input) {
  String output = "";
  LinkedHashMap < String, Integer > map = new LinkedHashMap<String, Integer>();
  StringTokenizer st = new StringTokenizer(data.toLowerCase());
  int i = 0;
  while (st.hasMoreTokens()) {
    String tmp = st.nextToken();
    map.put(tmp, ++i);
  }
  //Định dạng chuỗi trả về
  Iterator it = map.entrySet().iterator();
  while (it.hasNext()) {
    Map.Entry item = (Map.Entry)it.next();
    output += (it.hasNext()) ? item.getKey() + "_" : item.getKey();
  }
  return output;
}
```

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

//Dữ liệu trả về



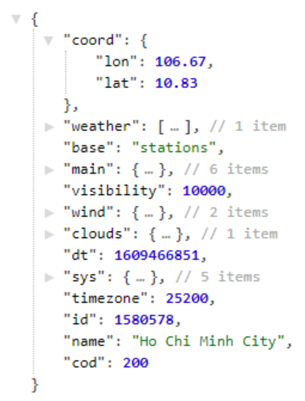
```
{
  "coord": { … }, // 2 items
  "weather": [
    {
      "id": 803,
      "main": "Clouds",
      "description": "broken clouds",
      "icon": "04d"
    }
  ],
  "base": "stations",
  "main": { … }, // 6 items
  "visibility": 10000,
  "wind": { … }, // 2 items
  "clouds": { … }, // 1 item
  "dt":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

```
// Lấy dữ liệu trong element weather:
import java.io.IOException;
import java.net.URL;
import java.util.Scanner;

import org.json.*; //add json library

private static String getDataWeather() {
  try {
    URL url = new URL("https://api.openweathermap.org/data/2.5/weather?q=SaiGon&appid=8a40b8cd501f5deec9801dd7e8e5585d");
    Scanner sc = new Scanner(url.openStream());
    String inline = "";
    while (sc.hasNext())
```

```
      inline += sc.nextLine();
    sc.close();

    JSONObject obj = new JSONObject(inline);
    JSONArray arr = (JSONArray) obj.get("weather");
    JSONObject objJSOn = (JSONObject) arr.get(0);
    return objJSOn.get("icon") + objJSOn.get("id") + objJSOn.get("main") + objJSOn.get("description");
  } catch (IOException e) {
    return "Lỗi hệ thống.";
  }
}
```

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

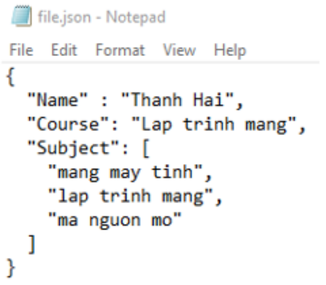
```
import java.io.IOException;
import java.net.URL;
import java.util.Scanner;

import org.json.*; //add json library

private static String getDataCoord() {
  try {
    URL url = new URL("https://api.openweathermap.org/data/2.5/weather?q=SaiGon&appid=8a40b8cd501f5deec9801dd7e8e5585d");
    Scanner sc = new Scanner(url.openStream());
    String inline = "";
    while (sc.hasNext())
      inline += sc.nextLine();
    sc.close();

    JSONObject obj = new JSONObject(inline);
    JSONObject objJSOn = (JSONObject) obj.get("coord");
    return objJSOn.get("lon") + " " + objJSOn.get("lat");
  } catch (IOException e) {
    return "Lỗi hệ thống.";
  }
}
```

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

```
import java.util.*;

public static String Cau1(String input) {
  try {
    StringTokenizer st = new StringTokenizer(input, "+-*/", true);
    float rs = Float.parseFloat(st.nextToken());
    while (st.hasMoreTokens()) {
      String next = st.nextToken();
      if (next.equals("+")) {
        rs += Float.parseFloat(st.nextToken());
      } else if (next.equals("-")) {
        rs -= Float.parseFloat(st.nextToken());
      } else if (next.equals("*")) {
        rs *= Float.parseFloat(st.nextToken());
      } else if (next.equals("/")) {
        rs /= Float.parseFloat(st.nextToken());
      } else {
        return "Định dạng không đúng. Cần kiểm tra lại.";
      }
    }
    return rs + "";
  } catch (NumberFormatException nfe) {
    return "Định dạng không đúng. Cần kiểm tra lại.";
  }
}
```

8. Đọc JSON từ file

// Cấu trúc file



```
{
  "Name" : "Thanh Hai",
  "Course": "Lap trinh mang",
  "Subject": [
    "mang may tinh",
    "lap trinh mang",
    "ma nguon mo"
  ]
}
```

// Xử lý

```
import java.io.FileReader;
import org.json.simple.*;
import org.json.simple.parser.*;

private static String getDataFile() {
  JSONParser parser = new JSONParser();
  try {
    Object obj = parser.parse(new FileReader("file.json"));
    JSONObject jsonObject = (JSONObject)obj;
    //Lấy value của name
    String name = (String)jsonObject.get("Name");
    //Lấy value của course
    String course = (String)jsonObject.get("Course");
    //Lấy value của subject
    JSONArray subject = (JSONArray)jsonObject.get("Subject");

    return name + " " + course + " " + subject;
  } catch (Exception e) {
    return "Lỗi hệ thống.";
  }
}
```

9. Tra từ trong từ điển

```
import java.io.File;
import java.util.*;

private String traTu(String input) {
  try {
    File myObj = new File("./src/txt.txt");
    Scanner scanner = new Scanner(myObj);

    HashMap < String, String > map1 = new HashMap<String, String>();
    HashMap < String, String > map2 = new HashMap<String, String>();

    while (scanner.hasNextLine()) {
      String data = scanner.nextLine();
      String[] tmp = data.split(";");
      map1.put(tmp[0], tmp[1]);
      map2.put(tmp[1], tmp[0]);
    }
    scanner.close();

    for (Map.Entry < String, String > entry : map1.entrySet()) {
      if (entry.getKey().equalsIgnoreCase(input)) {
        return entry.getValue();
      }
    }
    for (Map.Entry < String, String > entry : map2.entrySet()) {
      if (entry.getKey().equalsIgnoreCase(input)) {
        return entry.getValue();
      }
    }
  } catch (Exception e) {
    return "Lỗi hệ thống.";
  }
  return "Không tìm thấy từ trong từ điển 😢😢";
}
```

10. Đoán số

```
// Sinh số ngẫu nhiên từ 1 đến <= 100
private static int soNgauNhien = (int)(Math.random() * 100 + 1);

private static String doanSo(String input) {
  try {
    int so = Integer.parseInt(input);
    System.out.println("So ngau nhien la = " + soNgauNhien);
    if (so == soNgauNhien) {
      return "Chính xác.";
    } else if (so < soNgauNhien) {
      return "Cần gửi số lớn hơn.";
    } else {
      return "Cần gửi số bé hơn.";
    }
  } catch (NumberFormatException nfe) {
    return "Không phải là số.";
  }
}
```

11. Đảo ngược chuỗi

```
private static String daoNguocChuoi(String input) {
  String rs = "";
  String[] tmp = input.split(" ");
  for (int i = 0; i < tmp.length; i++) {
    rs += new StringBuilder(tmp[i]).reverse().toString();
    if (i == tmp.length - 1) continue;
    rs += " ";
  }
  return rs;
}
```

II. TCP

1. TCP Server

```java
import java.io.*;
import java.net.*;

public class Server {

  //Viết hàm Xử lí logic tại đây (Lưu ý: Trả về String)
  private String xuly(String input) {
    return "";
  }

  private Server(int port) {
    try {
      ServerSocket server = new ServerSocket(port);
      System.out.println("Server started");
      System.out.println("Waiting for a client...");
      Socket socket = server.accept();
      System.out.println("Client accepted");

      BufferedWriter out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
      BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

      String line = ""; // Nội dung Server nhận từ Client
      String dataSend = ""; // Nội dung Server gửi về client

      do {
        line = in.readLine(); //Nhận data từ client gửi
        dataSend = xuly(line); // Data gửi về client là nội dung hàm xử lý

        //Server gửi data về client
        out.write(dataSend);
        out.newLine();
        out.flush();
      } while (!line.equalsIgnoreCase("bye"));

      //CLOSE
      in.close();
      out.close();
      socket.close();
      server.close();
      System.err.println("Server closed");
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  public static void main(String[] args) {
    Server server = new Server(5000);
  }
}
```

2. TCP Client

```java
import java.io.*;
import java.net.Socket;

public class Client {
  private Client(String address, int port) {
    try {
      Socket socket = new Socket(address, port);
      System.out.println("Connect to server successful");

      BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
      BufferedWriter out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
      BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));

      String line = ""; // Nội dung Client gửi lên Server

      do {
        // Nhập data gửi lên Server
        System.out.print("Enter content: ");
        line = stdIn.readLine();
        if (line.equalsIgnoreCase("bye")) {
          break;
        }

        // Client gửi data lên Server
        out.write(line);
        out.newLine();
        out.flush();

        // Nhận data từ Server gửi về
        String dt = in.readLine();
        if (dt != null) {
          System.out.println("Ket qua: " + dt);
        }
      } while (!line.equalsIgnoreCase("bye"));

      //CLOSE
      System.out.print("See you later ❤️❤️");
      in.close();
      out.close();
      socket.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  public static void main(String[] args) {
    Client client1 = new Client("127.0.0.1", 5000);
  }
}
```

III. UDP

1. UDP Server

```
import java.io.IOException;
import java.net.*;

public class Server {
  private static int buffsize = 512;
  private static int port = 1234;

  private static String xuly(String input) {
    return "";
  }

  public static void main(String[] args) {
    DatagramSocket socket;
    DatagramPacket dpreceive, dpsend;
    try {
      socket = new DatagramSocket(port);
      dpreceive = new DatagramPacket(new byte[buffsize], buffsize);
      System.out.println("Server is ready!");
      while (true) {
        // Server nhận dữ liệu từ Client
        socket.receive(dpreceive);
        String dataReceived = new String(dpreceive.getData(), 0, dpreceive.getLength());
```

```
          System.out.println("Server received" + dataReceived);

          //catch keyword stop server
          if (dataReceived.equals("bye")) {
            System.out.println("Server closed!");
            socket.close();
            break;
          }

          String dataSend = xuly(dataReceived); // Nội dung Server gửi về Client

          // Server gửi data về Client
          dpsend = new DatagramPacket(dataSend.getBytes(), dataSend.getBytes().length, dpreceive.getAddress(), dpreceive.getPort());
          System.out.println("Server sent: " + dataSend);
          socket.send(dpsend);
        }
      } catch (IOException e) {
        System.err.println(e);
      }
    }
  }
```

2. UDP Client

```
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Scanner;

public class Client {

  private static int destPort = 1234;
  private static String hostname = "localhost";

  public static void main(String[] args) {
    DatagramSocket socket;
    DatagramPacket dpsend, dpreceive;
    InetAddress add; Scanner stdIn;
    System.out.println("Client is ready!");

    try {
      add = InetAddress.getByName(hostname);
      socket = new DatagramSocket();
      stdIn = new Scanner(System.in);

      while (true) {
        // Nhập nội dung gửi tới Server
        System.out.print("Client input: ");
        String dataSend = stdIn.nextLine();
        byte[] data = dataSend.getBytes();

        // Client gửi data tới Server
        dpsend = new DatagramPacket(data, data.length, add, destPort);
        System.out.println("Client sent: " + dataSend);
        socket.send(dpsend);

        //catch stop server
        if (dataSend.equals("bye")) {
          System.out.println("Client socket closed!");
          stdIn.close();
          socket.close();
          break;
        }

```

```
          // Client nhận data từ Server
          dpreceive = new DatagramPacket(new byte[512], 512);
          socket.receive(dpreceive);
          String dataReceived = new String(dpreceive.getData(), 0, dpreceive.getLength());
          System.out.println("Client get: " + dataReceived);
        }
      } catch (IOException e) { System.err.println(e); }
    }
  }
```

IV. Multi-Thread

1. Worker

```
import java.io.*;
import java.net.Socket;

public class Worker implements Runnable {
  private Socket socket;
  BufferedReader in;
  BufferedWriter out;

  private static String xuly(String input) {
    return "Kết quả nè: ";
  }

  public Worker(Socket s) throws IOException {
    this.socket = s;
    this.in = new BufferedReader(new InputStreamReader(s.getInputStream()));
    this.out = new BufferedWriter(new OutputStreamWriter(s.getOutputStream()));
  }

  public void run() {
    System.out.println("Client " + socket.toString() + " accepted");
    try {
      String input = "";
      while (true) {
        //Server nhận data từ Client
        input = in.readLine();
        System.out.println("Server received: " + input);
        if (input.equals("bye")) {
          break;
        }
        // Server gửi data về Client
        String output = xuly(input);
        out.write(output + '\n');
        out.flush();
        System.out.println("Server write: " + input);
      }
      System.out.println("Closed socket " + socket.toString());
            in.close();
      out.close();
      socket.close();
    } catch (IOException e) {
      System.out.println(e);
    }
  }
}
```

2. Server

```
import java.io.IOException;
import java.net.*;
import java.util.Vector;
```

```java
import java.util.concurrent.*;

public class Server {
  public static int port = 1234;
  public static int numThread = 2;
  private static ServerSocket server = null;
  public static Vector<Worker> workers = new Vector<>();

  public static void main(String[] args) throws IOException {
    ExecutorService executor = Executors.newFixedThreadPool(numThread);
    try {
      server = new ServerSocket(port);
      System.out.println("Server binding at port " + port);
      System.out.println("Waiting for client...");
      while (true) {
        Socket socket = server.accept();
        Worker client = new Worker(socket);
        executor.execute(client);
      }
    } catch (IOException e) {
      System.out.println(e);
    } finally {
      if (server != null)
        server.close();
    }
  }
}
```

3. Client

```java
import java.io.*;
import java.net.Socket;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

class SendMessage implements Runnable {
  private BufferedWriter out;
  private Socket socket;
  public SendMessage(Socket s, BufferedWriter o) {
    this.socket = s;
    this.out = o;
  }
  public void run() {
    try {
      while (true) {

        BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
        String data = stdIn.readLine();
        System.out.println("Input from client: " + data);
        out.write(data + '\n');
        out.flush();
        if (data.equals("bye"))
          break;
      }
      System.out.println("Client closed connection");
      out.close();
      socket.close();
    } catch (IOException e) { }
  }
}

class ReceiveMessage implements Runnable {
  private BufferedReader in;
  private Socket socket;
  public ReceiveMessage(Socket s, BufferedReader i) {
    this.socket = s;
    this.in = i;
  }
  public void run() {
    try {
      while (true) {
        String data = in.readLine();
        System.out.println("Receive: " + data);
      }
    } catch (IOException e) { }
  }
}

public class Client {
  private static String host = "localhost";
  private static int port = 1234;
  private static Socket socket;

  private static BufferedWriter out;
  private static BufferedReader in;

  public static void main(String[] args) throws IOException {
    socket = new Socket(host, port);
    System.out.println("Client connected");
    out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
          in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
    ExecutorService executor = Executors.newFixedThreadPool(2);
    SendMessage send = new SendMessage(socket, out);
    ReceiveMessage recv = new ReceiveMessage(socket, in);
    executor.execute(send);
    executor.execute(recv);
  }
}
```

V. Chat với người lạ

1. Interface

```
public interface Interface {
}

interface IListenerDisconnect{
    void onDisconnect();
}
interface  IListenerLogin{
    void onLoginSuccess();
    void onLoginFailed();
    void onConnectedAnotherClient();
    void onDisconnectedAnotherClient();
}
```

2. Worker

```
import java.io.*;
import java.net.Socket;

public class Worker implements Runnable {
    private String myName;
    private Socket socket;
    BufferedReader in;
    BufferedWriter out;
    private String stateConnect = "notLogin";//connected or waiting
    private String connectedUserName;
```

```java

    public Worker(Socket s, String name) throws Exception {
        this.socket = s;
        this.myName = name;
        this.in = new BufferedReader(new InputStreamReader(s.getInputStream()));
        this.out = new BufferedWriter(new OutputStreamWriter(s.getOutputStream()));
        this.connectedUserName = "";
    }
    //TODO: Failed login but connect to client
    public void run() {
        System.out.println("Client " + socket.toString() + " accepted");
        try {
            while (true) {
                String[] arrInput = null;
                String data = receiveDataFromClient();
                if (data.equals("bye")) {
                    send("bye");
                    if(stateConnect.equals("connected")) sendUniCast(connectedUserName, "disconnect");
                    break;
                }
                if (data.equals("disconnected"))
                {
                    stateConnect = "waiting";
                    send("Connected client disconnect!");
                }
                arrInput = data.split("#");//Validate input
                if (arrInput.length < 2 && stateConnect.equals("notLogin")) {
                    this.send("Du lieu khong hop le.");
                    continue;
                }

                switch (stateConnect) {
                    case "notLogin": {
                        handleNotLogin(arrInput);
                    }
                    break;
                    case "waiting": {
                        this.send("Waiting for another client ....");
                        handleWaiting();
                        //connect to waiting client
                    }
                    break;
                    case "connected": {
                        handleConnected(data);
                        //Send msg to connected client
                    }
                    break;
                }
            }
            //Client close connection
            System.out.println("Closed socket for client " + myName + " " + socket.toString());
            in.close();
            out.close();
            socket.close();
            Server.workers.remove(this);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }


    private void handleConnected(String data) {
        this.sendUniCast(this.connectedUserName, data);
    }

    public void handleNotLogin(String[] arrInput) {
            String optionalMessage = arrInput[0];
            String messageClient = arrInput[1];
            switch (optionalMessage) {
                case "all": {
                    if (sendBroadCast(messageClient)) {
                        this.send("Send message success!");
                    } else {
                        this.send("Send message error!");
                    }
                }
                break;
                case "login": {
                    if (validateLogin(messageClient)) {
                        this.myName = messageClient;
                        this.send("SuccessLogin");
                        this.stateConnect = "waiting";
                    } else {
                        this.send("Failed Login");
                        break;
                    }
                    // connect success
                    if (connectToAnotherClient()) {
                        this.send("Connected to " + connectedUserName);
                    } else {
                        this.send("Waiting for another client ......");
                    }
                }
                break;
                default: {
                    // Default send msg to another user
                    if (sendUniCast(optionalMessage, messageClient)) {
                        this.send("Send message success!");
                    } else {
                        this.send("Send message error");
                    }
                }
            }
        }

        public void handleWaiting() {
            while (true) {
                // connect success
                if (connectToAnotherClient()) {
                    this.send("Connected to " + connectedUserName);
                    break;
                }
            }
        }

        public boolean sendBroadCast(String input) {
            for (Worker worker : Server.workers) {
                if (!myName.equals(worker.myName)) {
                    if (!worker.send(input))
                        return false;
                }
            }
            return true;
        }

        public String receiveDataFromClient() {
            String input = "";
            try {
                input = in.readLine();
                System.out.println("Server received: " + input + " from " + socket.toString() + " # Client " + myName);
                return input;
            } catch (Exception e) {
                return "Exception\n";
            }
        }

        public boolean sendUniCast(String nameWorker, String input) {
            for (Worker worker : Server.workers) {
                if (nameWorker.equals(worker.myName)) {
                    if (worker.send(input)) return true;
                    else break;
                }
            }
            return false;
        }

        public boolean validateLogin(String userName) {
            for (Worker worker : Server.workers) {
                if (userName.equals(worker.myName))
                    return false; // userName exists
            }
            return true;
        }

        public boolean send(String data) {
            try {
                this.out.write(data + '\n');
                this.out.flush();
                return true;
            } catch (Exception e) {
                System.out.println(e + "ERROR Send msg to client ");
                return false;
            }
        }

        public boolean connectToAnotherClient() {
            for (Worker worker : Server.workers) {
                if (worker.stateConnect.equals("waiting")) {
                    // two same worker not connect
                    if (!worker.myName.equals(this.myName)) {
                        //add connectedUserName
                        worker.connectedUserName = this.myName;
                        this.connectedUserName = worker.myName;
                        //Change state
                        worker.stateConnect = "connected";
                        this.stateConnect = "connected";
                        //send announcement to client of worker connected
                        worker.send("Connected to " + worker.connectedUserName);
                        worker.send("Please enter message to send "+ worker.connectedUserName);
                        return true;
                    }
                }
            }
            return false;
        }
    }
```

3. Server

```
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Vector;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class Server {
    public static int port = 1234;
    public static int numThread = 4;
    private static ServerSocket server = null;
    public static Vector<Worker> workers = new Vector<>();

    public static void main(String[] args) throws Exception {
        int i = 0;
        ExecutorService executor = Executors.newFixedThreadPool(numThread);
        try {
            server = new ServerSocket(port);
            System.out.println("Server binding at port " + port);
            System.out.println("Waiting for client...");
            while(true) {
                i++;
                Socket socket = server.accept();
                Worker client = new Worker(socket, Integer.toString(i));
                workers.add(client);
                executor.execute(client);
            }
        } catch (Exception e) {
            System.out.println(e);
        } finally {
            if(server!=null)
                server.close();
        }
    }
}
```

4. Client

```
import java.io.*;
import java.net.Socket;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

class SendMessage implements Runnable {
    private BufferedWriter out;
    private Socket socket;
    public boolean isLogin = false;
    public String msgWorker = "";
    public SendMessage(Socket s, BufferedWriter o) {
        this.socket = s;
        this.out = o;
    }
    public void run() {
        System.out.println("Please enter your username: ");
        try {
            BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
            while(true) {
                String data = "";
                data += stdIn.readLine();
                if(msgWorker.equals("disconnected")) {
                    data = "disconnected";
                    msgWorker = "";
                }
                if(!isLogin){
                    System.out.println("Please enter your username: ");
                    String temp = data;
                    data = "login#"+temp;
                    System.out.println("Input from client: " + data);
                }else {
                    System.out.println("Your message: "+ data);
                }

                out.write(data+'\n');
                out.flush();
            if(data.equals("bye")) {
               stdIn.close();
               break;
            }
         }
         out.close();
      } catch (Exception e) {}
   }
}

class ReceiveMessage implements Runnable {
   private BufferedReader in;
   private Socket socket;
   private IListenerDisconnect iListenerDisconnect;
   private IListenerLogin iListenerLogin;
   public ReceiveMessage(Socket s, BufferedReader i, IListenerDisconnect iListenerDisconnect, IListenerLogin iListenerLogin) {
      this.socket = s;
      this.in = i;
      this.iListenerDisconnect = iListenerDisconnect;
      this.iListenerLogin = iListenerLogin;
   }
   public void run() {
      try {
         while(true) {
            String data = in.readLine();
            System.out.println("Receive: " + data);
            if(data.equals("bye")) {
               System.out.println("Exit program");
               iListenerDisconnect.onDisconnect();
               break;
            }
            if(data.equals("SuccessLogin")) iListenerLogin.onLoginSuccess();
            if(data.equals("disconnect")) iListenerLogin.onDisconnectedAnotherClient();
         }
         System.out.println("Close connection");
         in.close();
         socket.close();

      } catch (Exception e) {}
   }
}

public class Client {
   private static String host = "localhost";
   private static int port = 1234;
   private static Socket socket;
   private static BufferedWriter out;
   private static BufferedReader in;

   public static void main(String[] args) throws Exception {
      socket = new Socket(host, port);
      System.out.println("Client connected");
      out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
      in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
      ExecutorService executor = Executors.newFixedThreadPool(2);
      SendMessage send = new SendMessage(socket, out);

      ReceiveMessage recv = new ReceiveMessage(socket, in, () -> executor.shutdownNow(), new IListenerLogin() {
         @Override
         public void onLoginSuccess() {
            send.isLogin = true;
         }

         @Override
         public void onLoginFailed() {

            }

            @Override
            public void onConnectedAnotherClient() {
            }

            @Override
            public void onDisconnectedAnotherClient() {
                send.msgWorker = "disconnected";
            }
        });
        executor.execute(send);
        executor.execute(recv);

    }
}
```

VI. Chat multiclients

1. Worker

```
import java.io.*;
import java.net.Socket;
import java.util.regex.Pattern;

public class Worker implements Runnable {
  private String myName;
  private Socket socket;
  BufferedReader in;
  BufferedWriter out;

  public Worker(Socket s, String name) throws IOException {
    this.socket = s;
    this.myName = name;
    this.in = new BufferedReader(new InputStreamReader(s.getInputStream()));
    this.out = new BufferedWriter(new OutputStreamWriter(s.getOutputStream()));
  }

  public void run() {
    System.out.println("Client " + socket.toString() + " is accepted");
    try {
      sendUnicast(myName, "NAME#" + myName);
      sendBroadcast(myName + " is online");
      String input = "";
      while (true) {
        input = in.readLine();
        System.out.println("Server received: " + input + " from " + socket.toString() + " # Client " + myName);
        if (input.equals("bye")) {
          break;
        }
        else if (input.startsWith("all#")) {
          String msg = input.substring(4);
          out.write(sendBroadcast(msg));
          out.flush();
        } else if (checkInput(input.split("#")[0])) {
          String[] data = input.split("#");
          out.write(sendUnicast(data[0], input.substring(data[0].length() + 1)));
          out.flush();
        } else {
          out.write("Sai định dạng");
        }
      }
      System.out.println("Closed socket for client " + myName + " " + socket.toString());
      sendBroadcast(myName + " is offline");
      in.close();
```

```
        out.close();
        socket.close();
        removeWorker(myName);
      } catch (IOException e) {
        System.out.println(e);
      }
    }

    private boolean checkInput(String input) {
      Pattern pattern = Pattern.compile("^\\d+");
      return pattern.matcher(input).matches();
    }

    private String sendBroadcast(String msg){
      try {
        for (Worker worker : Server.workers) {
          if (!myName.equals(worker.myName)) {
            worker.out.write(msg + '\n');
            worker.out.flush();
          }
        }
      } catch (Exception e) {
        return "Can't send broadcast\n";
      }
      return "Sent broadcast successful\n";
    }

    private String sendUnicast(String name, String msg) throws IOException {
      try {
        for (Worker worker : Server.workers) {
          if (name.equals(worker.myName)) {
            worker.out.write(msg + '\n');
            worker.out.flush();
            return "Sent message successful\n";     // success
          }
        }
      } catch (Exception e) {
        return "Exception\n";
      } // exception
      return "Can't send broadcast\n"; // client not found
    }

    private boolean removeWorker(String name) {
      try {
        for (Worker worker : Server.workers)
        if (name.equals(worker.myName)) {
          Server.workers.remove(worker);
          break;
        }
      } catch (Exception e) {
        return false;
      }
      return true;
    }
  }
```

2. Server

```
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Vector;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

```

```
public class Server {
  public static int port = 1234;
  public static int numThread = 10;
  public static ServerSocket server = null;
  public static Vector<Worker> workers = new Vector<>();

  public static void main(String[] args) throws IOException {
    int i = 0;
    ExecutorService executor = Executors.newFixedThreadPool(numThread);
    try {
      server = new ServerSocket(port);
      System.out.println("Server binding at port " + port);
      System.out.println("Waiting for clients...");
      while (true) {
        i++;
        Socket socket = server.accept();
        Worker client = new Worker(socket, Integer.toString(i));
        workers.add(client);
        executor.execute(client);
      }
    } catch (IOException e) {
      System.out.println(e);
    } finally {
      if (server != null)
      server.close();
    }
  }
}
```

3. Client

```
import java.io.*;
import java.net.Socket;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

class SendMessage implements Runnable {
  private BufferedWriter out;
  private Socket socket;
  public SendMessage(Socket s, BufferedWriter o) {
    this.socket = s;
    this.out = o;
  }
  public void run() {
    try {
      while (true) {
        BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
        String data = stdIn.readLine();
        out.write(data + '\n');
        out.flush();
        if (data.equals("bye"))
          break;
      }
      System.out.println("CLIENT " + Client.myName + " closed connection");
      out.close();
      socket.close();
      Client.executor.shutdownNow();
    } catch (IOException e) { }
  }
}

class ReceiveMessage implements Runnable {
  private BufferedReader in;
  private Socket socket;
  public ReceiveMessage(Socket s, BufferedReader i) {
    this.socket = s;
    this.in = i;
  }
  public void run() {
    try {
      while (true) {
        String data = in.readLine();
        if (data.startsWith("NAME#")) {
          Client.myName = data.split("#")[1];
          System.out.println("CLIENT " + Client.myName + " connected");
          continue;
        }
        if (data.equals("bye"))
          break;
        System.out.println("CLIENT " + Client.myName + " received: " + data);
      }
      socket.close();
    } catch (IOException e) { }
  }
}

public class Client {
  public static ExecutorService executor;
  private static String host = "localhost";
  private static int port = 1234;
  private static Socket socket;
  public static String myName = "";

  private static BufferedWriter out;
  private static BufferedReader in;

  public static void main(String[] args) throws IOException, InterruptedException {
    socket = new Socket(host, port);
    out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
    in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
    executor = Executors.newFixedThreadPool(2);
    SendMessage send = new SendMessage(socket, out);
    ReceiveMessage recv = new ReceiveMessage(socket, in);
    executor.execute(send);
    executor.execute(recv);
  }
}
```