

Phần 1: CÁC BÀI TOÁN CƠ BẢN

Bài 1:

Viết chương trình thực hiện các phép tính căn bản (cộng, trừ, nhân, chia) với dữ liệu nhập vào dạng chuỗi phép tính (không có khoảng trắng). Nếu dữ liệu nhập vào không đúng định dạng hoặc không thuộc 1 trong 4 phép tính căn bản thì báo lỗi.

```
Input: 123+34
Output: 157

public class Bai1 {
    private static BufferedReader stdIn = null;

    public static double calculate(String in) {
        if(!in.matches("^\d{1,}[-*/]\d{1,}$")){
            System.out.println("Invalid syntax");
            return -1;
        }
        double res=0;
        StringTokenizer st = new StringTokenizer(in, "+-*/", true);
        while(st.hasMoreTokens()){
            String token = st.nextToken();
            if(!token.matches("[+-*/]")){
                res = Double.parseDouble(token);
            } else if (token.equals("/")){
                res /= Double.parseDouble(st.nextToken());
            } else if (token.equals("*")){
                res *= Double.parseDouble(st.nextToken());
            } else if (token.equals("+")){
                res += Double.parseDouble(st.nextToken());
            } else {
                res -= Double.parseDouble(st.nextToken());
            }
        }
        return res;
    }
    public static void main(String[] args) throws IOException{
        stdIn = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Input:");
        String line = stdIn.readLine();
        Double res = calculate(line);
        System.out.println("Output: "+res);
    }
}
```

Bài 2:

Viết chương trình nhập vào một chuỗi chứa một số từ lặp lại và giữa các từ có thể có 1 hoặc nhiều khoảng trắng, kết quả xuất ra chuỗi được định dạng chỉ chứa 1nỗi từ xuất hiện một lần (theo đúng thứ tự lần đầu xuất hiện) và chỉ có 1 khoảng trắng giữa các từ.

Input: " Dai hoc sai gon la mot trong nhung truong dai hoc lau doi nhat sai gon"

Output: "Dai hoc sai gon la mot trong nhung truong dai hoc lau doi nhat"

```
public class Bai2 {
    private static BufferedReader stdIn = null;

    public static String format(String in){
        StringTokenizer st = new StringTokenizer(in);
        ArrayList<String> tmp = new ArrayList<>();
        String token;
        while(st.hasMoreTokens()){
            token = st.nextToken().toLowerCase();
            if(!tmp.contains(token))
                tmp.add(token);
        }
        String res="";
        for(String s: tmp)
            res+=s+" ";
        return res;
    }

    public static void main(String[] args) throws IOException{
        stdIn = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Input:");
        String line = stdIn.readLine();
        System.out.println("Output: "+format(line));
    }
}
```

Bài 3:

Viết chương trình tra từ điển Anh-Việt, Việt-Anh đơn giản sử dụng cơ sở dữ liệu từ điển trong file dictionary.txt do sinh viên tự tạo. Cấu trúc file này gồm nhiều dòng, mỗi dòng chứa 1 từ tiếng Anh và từ tiếng Việt không dấu, phân cách nhau bởi kí tự;

Yêu cầu: nhập vào từ tiếng Anh sẽ trả về từ tiếng Việt và ngược lại. Trường hợp người dùng nhập vào một từ không có trong từ điển thì trả về thông báo lỗi.

```
public class Bai3 {
    private static BufferedReader stdIn = null;

    private static ArrayList<String> readFile() throws FileNotFoundException{
        ArrayList<String> res=new ArrayList<String>();
        // Đọc dữ liệu từ File với Scanner
        FileInputStream fileInputStream = new FileInputStream("dictionary.txt");
        Scanner scanner = new Scanner(fileInputStream);

        try {
            while (scanner.hasNextLine()) {
                res.add(scanner.nextLine());
            }
        } finally {
            try {
                scanner.close();
                fileInputStream.close();
            } catch (IOException ex) {
                System.err.println(ex);
            }
        }
        return res;
    }

    public static String translate(String in) throws FileNotFoundException{
        StringTokenizer st = new StringTokenizer(in, ";");
        ArrayList<String> dictionary= new ArrayList<String>();
        dictionary=readFile();

        for(String s: dictionary){
            String[] words=s.split(";");
            if(in.toLowerCase().equals(words[0].toLowerCase()))
                return words[1];
            else if(in.toLowerCase().equals(words[1].toLowerCase())))
                return words[0];
        }
        return "The word does not exist in the dictionary";
    }

    public static void main(String[] args) throws IOException{
        stdIn = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Input:");
        String line = stdIn.readLine();
        System.out.println("Output: "+translate(line));
    }
}
```

Phần 2: LẬP TRÌNH THAO TÁC VỚI ĐỊA CHỈ IP

Bài 1:

Viết chương trình nhận vào 1 domain name từ người dùng và xuất ra console IP tương ứng với domain name đó hoặc thông báo lỗi nếu domain name không tồn tại/không có IP tương ứng. Chương trình kết thúc khi người dùng nhập từ khóa **exit**.

```
Input:      sgu.edu.vn
Output:     124.158.134

public class Bai1 {
    private static BufferedReader stdIn = null;

    public static String getIP(String in) {
        String res = "";
        try {
            res = InetAddress.getByName(in).getHostAddress();
        } catch (UnknownHostException ex) {
            res = "Could not find "+in;
        }
        return res;
    }

    public static void main(String[] args) throws IOException{
        stdIn = new BufferedReader(new InputStreamReader(System.in));
        while(true){
            System.out.println("Input:");
            String line = stdIn.readLine();
            if(line.equals("exit"))
                break;
            System.out.println("Output: "+getIP(line));
        }
    }
}
```

Bài 2:

Viết chương trình đọc 1 file text chứa danh sách các domain name (mỗi domain name nằm trên 1 dòng) và xuất ra console danh sách IP tương ứng (hoặc thông báo lỗi nếu domain name không tồn tại/không có IP tương ứng).

Cấu trúc file input

```
sgu.edu.vn  
fit.sgu.edu.vn  
facebook.com  
svsgu.com  
tuoitre.vn  
thongtindaotao.sgu.edu.vn  
daotao1.sgu.edu.vn  
baomoi.com
```

Nội dung xuất ra trên console:

```
Domain name sgu.edu.vn has IP 124.158.13.4  
Domain name fit.sgu.edu.vn has IP 112.213.89.168  
Domain name facebook.com has IP 157.240.218.35  
Domain name svsgu.com is not valid  
Domain name tuoitre.vn has IP 222.255.239.80  
Domain name thongtindaotao.sgu.edu.vn has IP 210.211.117.37  
Domain name daotao1.sgu.edu.vn is not valid  
Domain name baomoi.com has IP 118.102.1.125
```

```
public class Bai2 {  
    private static ArrayList<String> readFile() throws FileNotFoundException{  
        ArrayList<String> res=new ArrayList<String>();  
        // Đọc dữ liệu từ File với Scanner  
        FileInputStream fileInputStream = new FileInputStream("domain.txt");  
        Scanner scanner = new Scanner(fileInputStream);  
  
        try {  
            while (scanner.hasNextLine()) {  
                res.add(scanner.nextLine());  
            }  
        } finally {  
            try {  
                scanner.close();  
                fileInputStream.close();  
            } catch (IOException ex) {  
                System.err.println(ex);  
            }  
        }  
        return res;  
    }  
}
```

```

public static void getIP() throws FileNotFoundException{
    ArrayList<String> domains= new ArrayList<>();
    domains = readFile();
    for(String s: domains){
        try {
            System.out.println("Domain name "+s+" has IP "
                +InetAddress.getByName(s).getHostAddress());
        } catch (UnknownHostException ex) {
            System.out.println("Domain name "+s+" is not valid");
        }
    }
}

public static void main(String[] args) throws FileNotFoundException{
    getIP();
}
}

```

Bài 3:

Viết chương trình đọc 1 file text chứa danh sách các IP (mỗi IP nằm trên 1 dòng) và xuất ra console danh sách IP kết nối hoặc không kết nối được.

Cấu trúc file input

```

124.158.13.4
112.213.89.168
157.240.218.35
124.158.13.5
118.226.90.138

```

Nội dung xuất ra trên console:

```

IP 124.158.13.4 is reachable
IP 112.213.89.168 is reachable
IP 157.240.218.35 is reachable
IP 124.158.13.5 is not reachable
IP 118.226.90.138 is not reachable
public class Bai3 {
    private static ArrayList<String> readFile() throws FileNotFoundException{
        ArrayList<String> res=new ArrayList<String>();
        // Đọc dữ liệu từ File với Scanner
        FileInputStream fileInputStream = new FileInputStream("IP.txt");
        Scanner scanner = new Scanner(fileInputStream);

        try {
            while (scanner.hasNextLine()) {
                res.add(scanner.nextLine());
            }
        } finally {
            try {
                scanner.close();
                fileInputStream.close();
            } catch (IOException ex) {
                System.err.println(ex);
            }
        }
        return res;
    }
}

```

```

public static void checkIP() throws FileNotFoundException, UnknownHostException{
    ArrayList<String> IPs= new ArrayList<>();
    IPs = readFile();
    boolean check = false;
    for(String s: IPs){
        try {
            InetAddress IP = InetAddress.getByName(s);
            check = IP.isReachable(1000);
        } catch (IOException ex) {
            System.out.println(ex);
        }
        if(check)
            System.out.println("IP "+s+" is reachable");
        else
            System.out.println("IP "+s+" is not reachable");
    }
}

public static void main(String[] args) throws FileNotFoundException, UnknownHostException{
    checkIP();
}
}

```

Bài 4:

Viết chương trình kiểm tra và liệt kê IP tất cả các thiết bị mạng đang online, thuộc cùng network với máy tính đang chạy chương trình.

Ví dụ IP của client: 192.168.1.10

Kết quả xuất ra: các IP thuộc đường mạng 192.168.1.x đang có thiết bị online

```

Checking IP 192.168.1.99
Checking IP 192.168.1.100 ==> 192.168.1.100 is online
Checking IP 192.168.1.101
Checking IP 192.168.1.102
Checking IP 192.168.1.103 ==> 192.168.1.103 is online
Checking IP 192.168.1.104
Checking IP 192.168.1.105
Checking IP 192.168.1.106 ==> 192.168.1.106 is online
Checking IP 192.168.1.107
Checking IP 192.168.1.108

```

```

public class Bai4 {
    public static void checkIP() throws UnknownHostException, IOException{
        InetAddress address = InetAddress.getLocalHost();
        String currentIP = address.getHostAddress();
        String ipWithNoFinalPart = currentIP.replaceAll("(.*\\.)\\\\d+$", "$1");
        System.out.println(ipWithNoFinalPart);

        for (int i = 1; i <= 255; i++) {
            String path = ipWithNoFinalPart.concat(String.valueOf(i));
            InetAddress subAdress = InetAddress.getByName(path);
            if (subAdress.isReachable(100)) {
                System.out.println("Checking IP" + subAdress + " ==> " + subAdress + " is online");
            } else {
                System.out.println("Checking IP" + subAdress);
            }
        }
    }

    public static void main(String[] args) throws IOException{
        checkIP();
    }
}

```

Phần 3: LẬP TRÌNH TCP SOCKET

```
public class Client {
    private static Socket socket = null;
    private static BufferedReader in = null;
    private static BufferedWriter out = null;
    private static BufferedReader stdIn = null;

    public static void main(String[] args) {
        try {
            socket = new Socket("localhost", 5000);
            in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
            stdIn = new BufferedReader(new InputStreamReader(System.in));
            while(true) {
                // Client nhận dữ liệu từ keyboard và gửi vào stream -> server
                System.out.print("Client input: ");
                String line = stdIn.readLine();
                out.write(line);
                out.newLine();
                out.flush();
                if(line.equals("bye"))
                    break;
                // Client nhận phản hồi từ server
                line = in.readLine();
                System.out.println("Client received: " + line);
            }
            System.out.println("Client closed connection");
            in.close();
            out.close();
            stdIn.close();
            socket.close();
        } catch (IOException e) { System.err.println(e); }
    }
}
```

Bài 1:

Viết chương trình gửi tin nhắn hai chiều giữa client-server sử dụng TCP socket.

- Client gửi 1 chuỗi ký tự bất kỳ đến server
 - Server nhận và gửi chuỗi đảo ngược về client
 - Client xuất kết quả ra console, chương trình kết thúc khi client gửi chuỗi **bye**.

```
public class Server {
    private static ServerSocket server = null;
    private static Socket socket = null;
    private static BufferedReader in = null;
    private static BufferedWriter out = null;

    public static void main(String[] args) {
        try {
            server = new ServerSocket(5000);
            System.out.println("Server started...");
            socket = server.accept();
            System.out.println("Client " + socket.getInetAddress() + " connected...");
            in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void close() {
        try {
            if (in != null)
                in.close();
            if (out != null)
                out.close();
            if (socket != null)
                socket.close();
            if (server != null)
                server.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```

        while(true) {
            // Server nhận dữ liệu từ client qua stream
            String line = in.readLine();
            if (line.equals("bye"))
                break;
            System.out.println("Server received: " + line);
            // Server gửi phản hồi ngược lại cho client (chuỗi đảo ngược)
            StringBuilder newline = new StringBuilder();
            newline.append(line);
            line = newline.reverse().toString();
            out.write(line);
            out.newLine();
            out.flush();
        }
        System.out.println("Server closed connection");
        // Đóng kết nối
        in.close();
        out.close();
        socket.close();
        server.close();
    } catch (IOException e) { System.err.println(e); }
}
}

```

Bài 2:

Viết chương trình tìm số hoàn hảo, hoạt động theo mô hình client-server, sử dụng TCP socket

- Client gửi 1 số n nguyên dương đến server.
- Server kiểm tra n, nếu:
 - Là số hoàn hảo: trả kết quả về client và xuất ra màn hình
 - Không phải số hoàn hảo: trả về client số hoàn hảo lớn hơn và gần n nhất.

```

public class Server {
    private static ServerSocket server = null;
    private static Socket socket = null;
    private static BufferedReader in = null;
    private static BufferedWriter out = null;

    private static boolean isPerfectNum(int num) {
        int sum=0;
        for(int i=1;i<=num/2;i++)
            if(num%i==0)
                sum+=i;
        return sum==num;
    }

    public static void main(String[] args) {
        try {
            server = new ServerSocket(5000);
            System.out.println("Server started...");
            socket = server.accept();
            System.out.println("Client " + socket.getInetAddress() + " connected...");
            in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
        }
    }
}

```

```

        while(true) {
            String line = in.readLine();
            if (line.equals("bye"))
                break;
            System.out.println("Server received: " + line);
            int number = Integer.parseInt(line);
            if(isPerfectNum(number)){
                line = String.valueOf(number) + " is perfect number";
            }
            else {
                boolean flag=false;
                while(!flag){
                    number++;
                    flag = isPerfectNum(number);
                }
                line = "The nearest perfect number is " + String.valueOf(number);
            }
            out.write(line);
            out.newLine();
            out.flush();
        }
        System.out.println("Server closed connection");
        in.close();
        out.close();
        socket.close();
        server.close();
    } catch (IOException e) { System.err.println(e); }
}
}

```

Bài 3:

Viết chương trình phân tích số, hoạt động theo mô hình client-server, sử dụng TCP socket

- Client gửi số nguyên dương $n \geq 10$ đến server.
- Server phân tích n thành tích các số nguyên tố và gửi trả ngược lại client
- Client xuất kết quả ra console

```

public class Server {
    private static ServerSocket server = null;
    private static Socket socket = null;
    private static BufferedReader input = null;
    private static BufferedWriter output = null;

    public static List<Integer> factorizePrime(int n) {
        int i = 2;
        List<Integer> listNumbers = new ArrayList<Integer>();
        while(n>1) {
            if(n%i==0) {
                n=n/i;
                listNumbers.add(i);
            }else{
                i++;
            }
        }
        if (listNumbers.isEmpty()) {
            listNumbers.add(n);
        }
        return listNumbers;
    }
}

```

```
public static void main(String[] args) {
    try {
        server = new ServerSocket(5000);
        System.out.println("Server started...");
        socket = server.accept();
        System.out.println("Client " + socket.getInetAddress() + " connected...");
        input = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        output = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
        while (true) {
            String line = input.readLine();
            if (line.equals("bye")) {
                break;
            }
            if(Integer.parseInt(line)<10){
                output.write("N >= 10");
                output.newLine();
                output.flush();
                continue;
            }
            System.out.println("Server received: " + line);
            int n = Integer.parseInt(line);
            List<Integer> listNumbers = factorizePrime(n);
            int size = listNumbers.size();
            String plus = "x";
            StringBuilder stringBuil = new StringBuilder();

            int i=0;
            while(i<size-1) {
                stringBuil.append(listNumbers.get(i));
                stringBuil.append(plus);
                i++;
            }
            stringBuil.append(listNumbers.get(i));
            String result = stringBuil.toString();
            output.write(result);
            output.newLine();
            output.flush();
        }
        System.out.println("Server closed connection");
        // Đóng kết nối
        input.close();
        output.close();
        socket.close();
        server.close();
    } catch (IOException e) {
        System.err.println(e);
    }
}
}
```

Bài 4:

Viết chương trình đoán số, hoạt động theo mô hình client-server, sử dụng TCP socket

- Khi client kết nối, server tạo sẵn 1 số nguyên ngẫu nhiên n <= 100
- Client đoán số do server tạo, nếu không đúng, server cần gợi ý bằng cách cho biết số client gửi lớn hơn hay nhỏ hơn n.
- Quá trình lặp liên tục cho tới khi client gửi đúng số = n. Server xuất các thống kê: số lần

```
client đoán, tổng thời gian đoán.  
public class Client {  
    private static Socket socket = null;  
    private static BufferedReader in = null;  
    private static BufferedWriter out = null;  
    private static BufferedReader stdIn = null;  
  
    public static void main(String[] args) {  
        try {  
            socket = new Socket("localhost", 5000);  
            in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
            out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));  
            stdIn = new BufferedReader(new InputStreamReader(System.in));  
            while(true) {  
                // Client nhận dữ liệu từ keyboard và gửi vào stream -> server  
                System.out.print("Client input: ");  
                String line = stdIn.readLine();  
                out.write(line);  
                out.newLine();  
                out.flush();  
                if(line.equals("bye"))  
                    break;  
                // Client nhận phản hồi từ server  
                line = in.readLine();  
                if(line.contains("Total")){  
                    System.out.println("Client receive: "+line);  
                    stdIn.close();  
                    socket.close();  
                    break;  
                }  
                System.out.println("Client received: " + line);  
            }  
            System.out.println("Client closed connection");  
            in.close();  
            out.close();  
            stdIn.close();  
            socket.close();  
        } catch (IOException e) { System.err.println(e); }  
    }  
}
```

```
public class Server {
    private static ServerSocket server = null;
    private static Socket socket = null;
    private static BufferedReader in = null;
    private static BufferedWriter out = null;

    public static void main(String[] args) {
        Random rand = new Random();
        int ranNum = rand.nextInt(101);
        int count = 0;
        boolean flag = false;
        try {
            server = new ServerSocket(5000);
            System.out.println("Server started...");
            socket = server.accept();
            System.out.println("Client " + socket.getInetAddress() + " connected...");
            in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
            long timeStart = System.currentTimeMillis();
            while(true) {
                count++;
                // Server nhận dữ liệu từ client qua stream
                String line = in.readLine();
                if (line.equals("bye"))
                    break;
                System.out.println("Server received: " + line);
                if(Integer.parseInt(line)==ranNum){
                    long timeEnd = System.currentTimeMillis();
                    long latency= timeEnd - timeStart;
                    line="Total: "+latency+" ms - Times: "+count;
                    flag = true;
                } else if(Integer.parseInt(line) < ranNum) {
                    line="Input < number";
                } else
                    line="Input > number";
                out.write(line);
                out.newLine();
                out.flush();
                if(flag) break;
            }
            System.out.println("Server closed connection");
            // Đóng kết nối
            in.close();
            out.close();
            socket.close();
            server.close();
        } catch (IOException e) { System.err.println(e); }
    }
}
```

Bài 5:

Viết chương trình tính toán, hoạt động theo mô hình client-server, sử dụng TCP socket

- Client gửi 1 chuỗi phép toán gồm 2 số và 1 trong 4 phép toán (+, -, *, /) đến server
- Server phân tích chuỗi, tính kết quả và trả lại client hoặc trả thông báo lỗi nếu chuỗi phép toán không đúng format.
- Client xuất kết quả ra console

```
public class Server {  
    private static ServerSocket server = null;  
    private static Socket socket = null;  
    private static BufferedReader in = null;  
    private static BufferedWriter out = null;  
  
    public static String calculate(String in){  
        if(!in.matches("^\\d{1,}[+*/]\\d{1,}$")){  
            return "Invalid syntax";  
        }  
        double res=0;  
        StringTokenizer st = new StringTokenizer(in, "+-*/", true);  
        while(st.hasMoreTokens()) {  
            String token = st.nextToken();  
            if(!token.matches("[+*/]")){  
                res = Double.parseDouble(token);  
            } else if (token.equals("/")){  
                res /= Double.parseDouble(st.nextToken());  
            } else if (token.equals("*")){  
                res *= Double.parseDouble(st.nextToken());  
            } else if (token.equals("+")){  
                res += Double.parseDouble(st.nextToken());  
            } else {  
                res -= Double.parseDouble(st.nextToken());  
            }  
        }  
        return String.valueOf(res);  
    }  
  
    public static void main(String[] args) {  
        try {  
            server = new ServerSocket(5000);  
            System.out.println("Server started...");  
            socket = server.accept();  
            System.out.println("Client " + socket.getInetAddress() + " connected...");  
            in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
            out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));  
            while(true) {  
                // Server nhận dữ liệu từ client qua stream  
                String line = in.readLine();  
                if (line.equals("bye"))  
                    break;  
                System.out.println("Server received: " + line);  
            }  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```

        out.write(calculate(line));
        out.newLine();
        out.flush();
    }
    System.out.println("Server closed connection");
    // Đóng kết nối
    in.close();
    out.close();
    socket.close();
    server.close();
} catch (IOException e) { System.err.println(e); }
}

}

```

Bài 6:

Viết chương trình tính số Pi theo phương pháp Monte Carlo, hoạt động theo mô hình client-server, sử dụng TCP socket

- Client gửi số N đến server (N là số nguyên dương, có giá trị từ 1.000.000 trở lên).
- Server phát sinh N điểm ngẫu nhiên và tính Pi theo phương pháp Monte Carlo (tham khảo <https://www.geeksforgeeks.org/estimating-value-pi-using-monte-carlo/>)
- Client nhận số Pi từ server và in ra màn hình kèm theo thời gian trễ (từ lúc client gửi N cho đến lúc nhận Pi).

```

public class Client {
    private static Socket socket = null;
    private static BufferedReader in = null;
    private static BufferedWriter out = null;
    private static BufferedReader stdIn = null;

    public static void main(String[] args) {
        try {
            socket = new Socket("localhost", 5000);
            in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
            stdIn = new BufferedReader(new InputStreamReader(System.in));
            while(true) {
                // Client nhận dữ liệu từ keyboard và gửi vào stream -> server
                System.out.print("Client input: ");
                String line = stdIn.readLine();
                long msSend = System.currentTimeMillis();
                out.write(line);
                out.newLine();
                out.flush();
                if(line.equals("bye"))
                    break;
                // Client nhận phản hồi từ server
                line = in.readLine();
                long msReceived = System.currentTimeMillis();
                long latency= msReceived - msSend;
                System.out.println("Client received: " + line+ " "+latency+"ms");
            }
        }
    }
}

```

```

        System.out.println("Client closed connection");
        in.close();
        out.close();
        stdIn.close();
        socket.close();
    } catch (IOException e) { System.err.println(e); }
}

}

public class Server {
    private static ServerSocket server = null;
    private static Socket socket = null;
    private static BufferedReader in = null;
    private static BufferedWriter out = null;

    public static String estimatePi(int INTERVAL){
        if(INTERVAL<1000000)
            return "Please input N >= 1000000";
        double rand_x, rand_y, origin_dist, pi=0;
        int circle_points = 0, square_points = 0;

        for (int i = 0; i < (INTERVAL); i++) {

            rand_x = Math.random()*2-1;
            rand_y = Math.random()*2-1;

            origin_dist = rand_x * rand_x + rand_y * rand_y;

            if (origin_dist <= 1)
                circle_points++;

            square_points++;

            pi = ((4.0 * circle_points) / square_points);
        }
        return String.valueOf(pi);
    }
    public static void main(String[] args) {
        try {
            server = new ServerSocket(5000);
            System.out.println("Server started...");
            socket = server.accept();
            System.out.println("Client " + socket.getInetAddress() + " connected...");
            in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
            while(true) {
                // Server nhận dữ liệu từ client qua stream
                String line = in.readLine();
                if (line.equals("bye"))
                    break;
                System.out.println("Server received: " + line);
                out.write((String.valueOf(estimatePi(Integer.parseInt(line)))));
                out.newLine();
                out.flush();
            }
        }
    }
}

```

Bài 7:

Viết chương trình tính toán, hoạt động theo mô hình client-server, sử dụng TCP socket

- Client gửi 1 chuỗi phép toán gồm nhiều số phân cách nhau bởi 1 trong 4 phép toán (+, -, *, /) đến server, giả sử chuỗi phép toán không chứa các dấu ngoặc. Ví dụ chuỗi phép toán sau: **12+34-56*78/4+14-17**
- Server phân tích chuỗi, tính kết quả và trả lại client hoặc trả thông báo lỗi nếu chuỗi phép toán không đúng format.
- Client xuất kết quả ra console

```
public class Server {  
    private static ServerSocket server = null;  
    private static Socket socket = null;  
    private static BufferedReader in = null;  
    private static BufferedWriter out = null;  
  
    public static List<String> infixToPostfixConvert(String input) {  
        int priority = 0;  
        String postfixBuffer = "";  
        Stack<Character> stack = new Stack<Character>();  
        List<String> postfixArray = new ArrayList<String>();  
  
        for (int i = 0; i < input.length(); i++) {  
            char ch = input.charAt(i);  
            if (ch == '+' || ch == '-' || ch == '*' || ch == '/') {  
  
                if (postfixBuffer.length() > 0) {  
                    postfixArray.add(postfixBuffer);  
                }  
                postfixBuffer = "";  
                // check the precedence  
                if (stack.size() <= 0)  
                    stack.push(ch);  
                else {  
                    Character chTop = (Character) stack.peek();  
                    if (chTop == '*' || chTop == '/')  
                        priority = 1;  
                    else  
                        priority = 0;  
                    if (priority == 1) {  
                        if (ch == '+' || ch == '-') {  
                            postfixArray.add(String.valueOf(stack.pop()));  
                            i--;  
                        } else { // Same  
                            postfixArray.add(String.valueOf(stack.pop()));  
                            i--;  
                        }  
                    } else {  
                        if (ch == '+' || ch == '-') {  
                            postfixArray.add(String.valueOf(stack.pop()));  
                            stack.push(ch);  
                        } else  
                            stack.push(ch);  
                    }  
                }  
            }  
        }  
    }  
}
```

```
        } else {
            postfixBuffer += ch;
        }
    }
postfixArray.add(postfixBuffer);
int len = stack.size();
for (int j = 0; j < len; j++)
    postfixArray.add(stack.pop().toString());

return postfixArray;
}
public static double evaluatePostfix(String exp) {
    Stack<Double> stack = new Stack<>();

    for (int i = 0; i < exp.length(); i++) {
        char c = exp.charAt(i);

        if (c == ' ')
            continue;

        else if (Character.isDigit(c)) {
            int n = 0;

            while (Character.isDigit(c)) {
                n = n * 10 + (int) (c - '0');
                i++;
                c = exp.charAt(i);
            }
            i--;

            stack.push(Double.parseDouble(String.valueOf(n)));
        }

        else {
            double val1 = stack.pop();
            double val2 = stack.pop();

            switch (c) {
                case '+':
                    stack.push(val2 + val1);
                    break;

                case '-':
                    stack.push(val2 - val1);
                    break;

                case '/':
                    stack.push(val2 / val1);
                    break;
            }
        }
    }
    return stack.pop();
}
```

```

        case '*':
            stack.push(val2 * val1);
            break;
    }
}
return stack.pop();
}

public static void main(String[] args) {
    try {
        server = new ServerSocket(5000);
        System.out.println("Server started...");
        socket = server.accept();
        System.out.println("Client " + socket.getInetAddress() + " connected...");
        in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));

        while (true) {
            String line = in.readLine();
            if (line.equals("bye"))
                break;
            if(!line.matches("^\\d[-*/]\\d([-*/]\\d){0,}"))
                line = "Invalid syntax";
            else{
                List<String> exp = Server.infixToPostfixConvert(line);
                String input = "";
                for (String s : exp) {
                    input += " " + s;
                }

                out.write(line);
                out.newLine();
                out.flush();
            }
        }
        System.out.println("Server closed connection");
        in.close();
        out.close();
        socket.close();
        server.close();
    } catch (IOException e) {
        System.err.println(e);
    }
}
}

```

Phần 4: LẬP TRÌNH UDP SOCKET

```
public class Client {
    public static int destPort=1234;
    public static String hostName="localhost";
    public static void main(String[] args){
        DatagramSocket socket;
        DatagramPacket dpReceive, dpSend;
        InetAddress add; Scanner stdIn;
        try{
            add=InetAddress.getByName(hostName);
            socket=new DatagramSocket();
            stdIn=new Scanner(System.in);
            while(true){
                System.out.print("Client input: ");
                String tmp=stdIn.nextLine();
                byte[] data=tmp.getBytes();
                dpSend=new DatagramPacket(data, data.length, add, destPort);
                socket.send(dpSend);
                if(tmp.equals("bye")){
                    System.out.println("Client socket closed");
                    stdIn.close();
                    socket.close();
                    break;
                }
                dpReceive=new DatagramPacket(new byte [512], 512);
                socket.receive(dpReceive);
                tmp=new String(dpReceive.getData(),0,dpReceive.getLength());
                System.out.println("Client receive: "+tmp);
            }
        }catch(IOException e) {System.err.print(e);}
    }
}
```

Bài 1:

Viết chương trình gửi tin nhắn hai chiều giữa client-server sử dụng UDP socket.

- Client gửi 1 chuỗi ký tự bất kỳ đến server
- Server nhận và gửi chuỗi đảo ngược từng từ về client.
- Client xuất kết quả ra console, chương trình kết thúc khi client gửi chuỗi **bye**.

VD: client gửi chuỗi **hello world**, server trả lại chuỗi **olleh dlrow**

```
public class Server {  
    public static int bufferSize=512;  
    public static int port=1234;  
    public static DatagramSocket socket;  
    public static DatagramPacket dpReceive, dpSend;  
  
    public static void main(String[] args){  
        try{  
            System.out.print("Server: ");  
            socket=new DatagramSocket(1234);  
            dpReceive=new DatagramPacket(new byte [bufferSize], bufferSize);  
            while(true){  
                socket.receive(dpReceive);  
                String tmp=new String(dpReceive.getData(),0,dpReceive.getLength());  
                System.out.print(tmp);  
                if(tmp.equals("bye")){  
                    socket.close();  
                    break;  
                }  
                StringBuilder newline = new StringBuilder();  
                newline.append(tmp);  
                tmp = newline.reverse().toString();  
                dpSend = new DatagramPacket(tmp.getBytes(), tmp.getBytes().length,  
                dpReceive.getAddress(), dpReceive.getPort());  
                socket.send(dpSend);  
            }  
        }catch(IOException e) {System.err.print(e);}  
    }  
}
```

Bài 2:

Viết chương trình tìm số hoạt động theo mô hình client-server, sử dụng UDP socket

- Client gửi 1 số n nguyên dương đến server.
- Server đọc file data.txt (đã chuẩn bị sẵn, có ít nhất 100.000 số ngẫu nhiên, mỗi số có giá trị từ 0 → 100000) và đếm số lượng số có tổng các chữ số cộng lại bằng đúng n, trả kết quả về client.

```
public class Server {  
    public static int bufferSize=512;  
    public static int port=1234;  
    public static DatagramSocket socket;  
    public static DatagramPacket dpReceive, dpSend;  
  
    private static ArrayList<Integer> readFile() throws FileNotFoundException{  
        ArrayList<Integer> res=new ArrayList<>();  
        // Đọc dữ liệu từ File với Scanner  
        FileInputStream fileInputStream = new FileInputStream("data.txt");  
        Scanner scanner = new Scanner(fileInputStream);  
  
        try {  
            while (scanner.hasNextLine()) {  
                res.add(Integer.parseInt(scanner.nextLine()));  
            }  
        } finally {  
            try {  
                scanner.close();  
                fileInputStream.close();  
            } catch (IOException ex) {  
                System.err.println(ex);  
            }  
        }  
        return res;  
    }  
  
    public static int total(int n) {  
        int res = 0;  
        do {  
            res = res + n % 10;  
            n = n / 10;  
        } while (n > 0);  
        return res;  
    }  
  
    public static int count(int n) throws FileNotFoundException{  
        int res=0;  
        ArrayList<Integer> num = new ArrayList<>();  
        num = readFile();  
        for(int i: num){  
            if(total(i)==n)  
                res++;  
        }  
        return res;  
    }  
}
```

```
public static void main(String[] args) throws IOException{
    try{
        System.out.print("Server: ");
        socket=new DatagramSocket(1234);
        dpReceive=new DatagramPacket(new byte [bufferSize], bufferSize);
        while(true){
            socket.receive(dpReceive);
            String tmp=new String(dpReceive.getData(),0,dpReceive.getLength());
            System.out.print(tmp);
            if(tmp.equals("bye")){
                socket.close();
                break;
            }
            tmp = String.valueOf(count(Integer.parseInt(tmp)));
            dpSend = new DatagramPacket(tmp.getBytes(), tmp.getBytes().length,
            dpReceive.getAddress(), dpReceive.getPort());
            socket.send(dpSend);
        }

    }catch(IOException e) {System.err.print(e);}
}
}
```

Bài 4:

Viết chương trình đoán số, hoạt động theo mô hình client-server, sử dụng UDP socket

- Khi client kết nối, server tạo sẵn 1 số nguyên ngẫu nhiên n <= 500
- Client đoán số do server tạo, nếu không đúng, server cần gợi ý bằng cách cho biết số client gửi lớn hơn hay nhỏ hơn n.
- Quá trình lặp liên tục cho tới khi client gửi đúng số = n. Server xuất các thống kê: số lần client đoán, tổng thời gian đoán.

```
public class Client {  
    public static int destPort=1234;  
    public static String hostName="localhost";  
    public static void main(String[] args){  
        DatagramSocket socket;  
        DatagramPacket dpReceive, dpSend;  
        InetAddress add; Scanner stdIn;  
        try{  
            add=InetAddress.getByName(hostName);  
            socket=new DatagramSocket();  
            stdIn=new Scanner(System.in);  
            while(true){  
                System.out.print("Client input: ");  
                String tmp=stdIn.nextLine();  
                byte[] data=tmp.getBytes();  
                dpSend=new DatagramPacket(data, data.length, add, destPort);  
                socket.send(dpSend);  
                dpReceive=new DatagramPacket(new byte [512], 512);  
                socket.receive(dpReceive);  
                tmp=new String(dpReceive.getData(), 0, dpReceive.getLength());  
                if(tmp.contains("Total")){  
                    System.out.println("Client receive: "+tmp);  
                    stdIn.close();  
                    socket.close();  
                    break;  
                }  
                System.out.println("Client receive: "+tmp);  
            }  
        }catch(IOException e) {System.err.print(e);}  
    }  
}
```

```
public class Server {
    public static int bufferSize=512;
    public static int port=1234;
    public static DatagramSocket socket;
    public static DatagramPacket dpReceive, dpSend;

    public static void main(String[] args){
        Random rand = new Random();
        int ranNum = rand.nextInt(501);
        int count = 0;
        boolean flag = false;
        try{
            System.out.print("Server: ");
            socket=new DatagramSocket(1234);
            dpReceive=new DatagramPacket(new byte [bufferSize], bufferSize);
            long timeStart = System.currentTimeMillis();
            while(true){
                count++;
                socket.receive(dpReceive);
                String tmp=new String(dpReceive.getData(),0,dpReceive.getLength());
                if(Integer.parseInt(tmp)==ranNum){
                    long timeEnd = System.currentTimeMillis();
                    long latency= timeEnd - timeStart;
                    tmp="Total: "+latency+" ms - Times: "+count;
                    flag = true;
                } else if(Integer.parseInt(tmp) < ranNum) {
                    tmp="Input < number";
                } else
                    tmp="Input > number";
                dpSend = new DatagramPacket(tmp.getBytes(), tmp.getBytes().length,
                dpReceive.getAddress(), dpReceive.getPort());
                socket.send(dpSend);
                if(flag) break;
            }
        }catch(IOException e) {System.err.print(e);}
    }
}
```

Phần 5: LẬP TRÌNH MULTITHREAD

Bài 1:

Viết chương trình tính số Pi bằng phương pháp Monte Carlo, sử dụng multithread:

- Thread 1 & 2: mỗi thread phát sinh 1.000.000 points và đếm số point trong đường tròn.
Hai thread chạy cùng lúc, cùng cập nhật số point trong trường tròn vào 1 biến toàn cục chung.
- Sau khi thread 1 & 2 hoàn tất, main thread tính toán giá trị Pi dựa trên tổng số điểm trong đường tròn và tổng số points phát sinh, theo phương pháp Monte Carlo.

```
public class Bai1 {  
    // Biến toàn cục chung để lưu tổng số points trong đường tròn  
    private static AtomicInteger totalInCircle = new AtomicInteger(0);  
  
    // Hỗn xác định xem một point có nằm trong đường tròn hay không  
    private static boolean isInCircle(double x, double y) {  
        return x * x + y * y <= 1;  
    }  
  
    // Lớp chứa hàm phát sinh và đếm số points trong đường tròn  
    // Số points được tăng lên biến toàn cục total_in_circle  
    public static class PointsGenerator implements Runnable {  
        private int numPoints;  
  
        public PointsGenerator(int numPoints) {  
            this.numPoints = numPoints;  
        }  
  
        @Override  
        public void run() {  
            for (int i = 0; i < numPoints; i++) {  
                double x = Math.random() * 2 - 1;  
                double y = Math.random() * 2 - 1;  
                if (isInCircle(x, y)) {  
                    totalInCircle.incrementAndGet();  
                }  
            }  
        }  
    }  
    public static void main(String[] args) throws InterruptedException {  
        // Khởi tạo 2 thread và chạy cùng lúc  
        Thread t1 = new Thread(new PointsGenerator(1000000));  
        Thread t2 = new Thread(new PointsGenerator(1000000));  
        t1.start();  
        t2.start();  
  
        // Đợi cho 2 thread hoàn tất  
        t1.join();  
        t2.join();  
  
        // Tính toán và in ra kết quả  
        double pi = 4.0 * totalInCircle.get() / (1000000 * 2);  
        System.out.println("Giá trị Pi dựa trên phương pháp Monte Carlo: " + pi);  
    }  
}
```

Bài 2:

Viết chương trình đọc file data.txt chứa tối thiểu 1.000.000 số nguyên dương có giá trị ngẫu nhiên, sắp xếp các số tăng dần và xuất ra file out.txt

- Main thread: đọc file data.txt và phân chia số cho các thread sắp xếp.
- Thread 1, 2 & 3 **chạy cùng lúc**, mỗi thread **sắp xếp 1/3 số lượng số**.
- Main thread đợi thread 1, 2 & 3 hoàn tất, ghi dữ liệu vào file out.txt. Đồng thời cho biết tổng thời gian chạy chương trình.

Viết lại chương trình trên sử dụng single thread và tính tổng thời gian chạy chương trình, đổi chiều kết quả với chương trình multithread.

```
public class Bai2 {  
    // Hàm sắp xếp một mảng số nguyên dương  
    private static int[] sort(int[] arr) {  
        Arrays.sort(arr);  
        return arr;  
    }  
    // Lớp chứa hàm sắp xếp một phần của mảng số  
    public static class SortTask implements Runnable {  
        private int[] arr;  
        private int start;  
        private int end;  
  
        public SortTask(int[] arr, int start, int end) {  
            this.arr = arr;  
            this.start = start;  
            this.end = end;  
        }  
        @Override  
        public void run() {  
            int[] sorted = sort(Arrays.copyOfRange(arr, start, end));  
            System.arraycopy(sorted, 0, arr, start, end - start);  
        }  
    }  
    public static void main(String[] args) throws IOException, IOException, InterruptedException {  
        // Đọc dữ liệu từ file data.txt  
        long b = System.currentTimeMillis();  
        BufferedReader reader = new BufferedReader(new FileReader("data.txt"));  
        List<Integer> data = new ArrayList<>();  
        String line;  
        while ((line = reader.readLine()) != null) {  
            data.add(Integer.parseInt(line));  
        }  
        reader.close();  
        // Chuyển dữ liệu thành mảng số nguyên và sắp xếp  
        int[] arr = sort(data.stream().mapToInt(i -> i).toArray());  
  
        // tạo 4 thread sắp xếp  
        int chunkSize = arr.length/3;  
        SortTask sortTask1= new SortTask(arr,0, chunkSize);  
        SortTask sortTask2= new SortTask(arr,chunkSize+1, chunkSize*2);  
        SortTask sortTask3= new SortTask(arr,chunkSize*2+1,arr.length);  
    }  
}
```

```
Thread t1= new Thread(sortTask1 );
Thread t2= new Thread(sortTask2 );
Thread t3= new Thread(sortTask3 );
t1.start();
t2.start();
t3.start();

t1.join();
t2.join();
t3.join();

// Ghi kết quả vào file out.txt
BufferedWriter writer = new BufferedWriter(new FileWriter("out.txt"));
for (int i : arr) {
    writer.write(String.valueOf(i));
    writer.newLine();
}
writer.close();
b = System.currentTimeMillis() - b;
System.out.println("Total time: "+b+ " miliseconds");
}

}
```

Bài 3:

Viết chương trình chat giữa 1 server và nhiều client theo mô hình client/server, sử dụng giao thức TCP và dùng cơ chế multithread:

- Main thread trên server mở sẵn port 1234 và chờ các client kết nối.
- Khi 1 client kết nối, server tạo ra 1 thread tương ứng (sử dụng 1 port ngẫu nhiên) để giao tiếp với client. Chuỗi dữ liệu do client gửi lên sẽ được server trả ngược về dưới dạng IN HOA. Quá trình giao tiếp giữa cặp client - server này sẽ kết thúc khi client gửi chuỗi bye.
- Main thread sẽ chỉ kết thúc khi người dùng chủ động dừng chương trình (hoặc sau 10 phút mà không có bất kỳ client nào kết nối - SV tự cài đặt)

```
public class Client {  
    private static Socket socket = null;  
    private static BufferedReader in = null;  
    private static BufferedWriter out = null;  
    private static BufferedReader stdIn = null;  
  
    public static void main(String[] args) {  
        try {  
            socket = new Socket("localhost", 1234);  
            in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
            out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));  
            stdIn = new BufferedReader(new InputStreamReader(System.in));  
            while(true) {  
                // Client nhận dữ liệu từ keyboard và gửi vào stream -> server  
                System.out.print("Client input: ");  
                String line = stdIn.readLine();  
                out.write(line);  
                out.newLine();  
                out.flush();  
                if(line.equals("bye"))  
                    break;  
                // Client nhận phản hồi từ server  
                line = in.readLine();  
                System.out.println("Client received: " + line);  
            }  
            System.out.println("Client closed connection");  
            in.close();  
            out.close();  
            stdIn.close();  
            socket.close();  
        } catch (IOException e) { System.err.println(e); }  
    }  
}
```

```

public class ClientHandler implements Runnable {
    private Socket socket;
    private String clientID;
    BufferedReader in;
    BufferedWriter out;

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

    public void handleSendData(Vector<ClientHandler> clientList, String inputData, String clientID) throws IOException {
        int error = 0;
        for (ClientHandler client : clientList) {
            if(Objects.equals(clientID, client.clientID)){
                if(Objects.equals(inputData, "bye")){
                    client.out.write("User " + clientID + " has been disconnected \n");
                    client.out.flush();
                    Server.clientList.remove(this);
                    Server.executor.remove(this);
                }
                else {
                    System.out.println("vao day: "+inputData.toLowerCase());
                    client.out.write(inputData.toUpperCase() +"\n");
                    client.out.flush();
                }
            }
        }
    }

    public void run() {
        System.out.println("Client " + socket.toString() + " accepted");
        Set<Thread> threadSet = Thread.getAllStackTraces().keySet();
        System.out.println(threadSet);
        try {
            while (true) {
                String inputData = in.readLine();
                //Handle if user's input is bye
                if (inputData.equals("bye")) {
                    handleSendData(Server.clientList, inputData,clientID);
                    System.out.println("Name " + clientID);
                    Server.clientList.remove(this);
                    System.out.println("Test cleint list");
                    System.out.println(Server.clientList);
                    Server.executor.remove(this);
                    break;
                }
                else {
                    handleSendData(Server.clientList, inputData,clientID);
                }
            }
            System.out.println("Closed socket for client " + socket.toString());
            in.close();
            out.close();
            socket.close();
            System.out.println(Server.executor.remove(this));
            System.out.println(threadSet);
        } catch (IOException e) {
            System.out.println(e);
        }
    }
}

```

```
public class Server {
    public static int port = 1234;
    public static int numThread = 10;
    public static Vector<ClientHandler> clientList = new Vector<>();
    public static ThreadPoolExecutor executor = (ThreadPoolExecutor) Executors.newFixedThreadPool(numThread);
    public static ScheduledExecutorService schedule = Executors.newScheduledThreadPool(1);
    public static CheckTime ct = new CheckTime();
    public static ServerSocket server;

    public static void main(String[] args) throws IOException {
        try {
            server = new ServerSocket(port);
            System.out.println("Server binding at port " + port);
            System.out.println("Waiting for client...");
            int i = 1;
            ScheduledFuture<?> scheduledFuture = schedule.scheduleAtFixedRate(ct, 600, 1, TimeUnit.SECONDS);
            while (true) {
                Socket socket = server.accept();
                ClientHandler client = new ClientHandler(socket, Integer.toString(i++));
                clientList.add(client);
                executor.execute(client);
                scheduledFuture.cancel(true);
            }
        } catch (IOException e) {
            System.out.println(e);
        }
        finally {
            if (server != null) {
                server.close();
            }
        }
    }
}

class CheckTime implements Runnable {
    @Override
    public void run() {
        System.out.println("stop time. Time: " + Instant.now());
        System.exit(0);
    }
}
```

Bài 4:

Viết chương trình chat giữa client 1 và client 2 thông qua server, theo mô hình client/server, sử dụng giao thức TCP và dùng cơ chế multithread:

- Main thread trên server mở sẵn port 1235 và chờ client 1, client 2 kết nối.
- Mỗi khi 1 client kết nối, server tạo 1 thread tương ứng để phục vụ client. Nếu:
 - Mới chỉ có 1 client kết nối, client đó sẽ ở trạng thái chờ client còn lại.
 - Có thêm client thứ hai kết nối, server lập tức gửi tin nhắn báo hiệu đến client kia và 2 client sẽ chat được qua lại (tin nhắn trao đổi giữa 2 client phải thông qua server)
- Khi có 1 client kết thúc bằng cách gửi chuỗi bye, server sẽ gửi thông báo kết thúc chat đến client còn lại và kết thúc cả 2 thread.
- Main thread sẽ chỉ kết thúc khi người dùng chủ động dừng chương trình (hoặc sau 10 phút mà không có bất kỳ client nào kết nối - SV tự cài đặt)

```
class Send implements Runnable {  
    private Socket socket;  
    private BufferedWriter out;  
  
    public Send(Socket s, BufferedWriter o, String bye) {  
        this.socket = s;  
        this.out = o;  
  
    }  
  
    public void run() {  
        BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));  
        try {  
            while (true) {  
                System.out.println("Input: ");  
                String data = stdIn.readLine();  
                out.write(data + "\n");  
                out.flush();  
                if (data.equals("bye")) {  
                    break;  
                }  
            }  
            this.socket.close();  
        } catch (IOException e) {  
            System.out.println(e.getMessage());  
        }  
    }  
}  
class Receive implements Runnable {  
    private Socket socket;  
    private BufferedReader in;  
  
    public Receive(Socket s, BufferedReader r) {  
        this.socket = s;  
        this.in = r;  
    }  
}
```

```
public void run() {
    try {
        while (true) {
            String data = in.readLine();
            System.out.println("Received: " + data);
        }
    } catch (IOException e) {
        System.out.println(e.getMessage());
    }
}

public class Client {

    private static String host = "localhost";
    private static int port = 1234;
    private static Socket socket;

    public static void main(String[] args) throws IOException {
        try {
            socket = new Socket(host, port);
            System.out.println("Client connected");
            BufferedWriter out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
            BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

            Send send = new Send(socket, out, "bye");
            Receive recv = new Receive(socket, in);
            ExecutorService excutor = Executors.newFixedThreadPool(2);
            excutor.execute(send);
            excutor.execute(recv);

        } catch (IOException e) {
            System.out.println(e);
        }
    }
}
```

```

public class ClientHandler implements Runnable {
    private Socket socket;
    private String name;
    BufferedReader in;
    BufferedWriter out;

    public ClientHandler(Socket s, String n) throws IOException {
        this.name = n;
        this.socket = s;
        in = new BufferedReader(new InputStreamReader(s.getInputStream()));
        out = new BufferedWriter(new OutputStreamWriter(s.getOutputStream()));
    }
    public void handleSendDataID(Vector<ClientHandler> clientList, String inputId, String inputData) throws IOException
    int error = 0;
    for (ClientHandler client : clientList) {
        //Nếu người dùng nhập exit
        if (inputId.equals("") && inputData.equals("")) {
            if (!name.equals(client.name)) {
                client.out.write("User " + name + " has been disconnected \n");
                client.out.flush();
            }
        }
        //Nếu người dùng bắt đầu với cú pháp all
        if (inputId.equals("all")) {
            if (!name.equals(client.name)) {
                client.out.write(inputData + "\n");
                client.out.flush();
            }
        } else {
            //Nếu người dùng bắt đầu với cú pháp có id
            if (client.name.equals(inputId)) {
                client.out.write(inputData + "\n"); client.out.flush();
                System.out.println("Server sent '" + inputData + "' from Client " + name + "--> Client " + inputId);
            } else {
                if (name.equals(client.name)) {
                    //Nếu người dùng nhập id không tồn tại
                    if (Integer.parseInt(inputId) > clientList.size()) {
                        client.out.write("User doesn't exist \n");
                        client.out.flush();
                    }
                }
            }
        }
    }
}
public void handleSendData (Vector<ClientHandler> clientList, String inputData) throws IOException {
    for (ClientHandler client : clientList) {
        if (!name.equals(client.name)) {
            client.out.write(inputData + "\n");
            client.out.flush();
        }
    }
}
public void run() {
    System.out.println("Client " + socket.toString() + " accepted");
    Set<Thread> threadSet = Thread.getAllStackTraces().keySet();
    System.out.println(threadSet);
    try {
        String inputData = null;
        while (true) {
            String input = in.readLine();
            if (input.equals("bye")) {
                handleSendData(Server.clientList,"");
                System.out.println("Name " + name);
                Server.clientList.remove(this);
            }
        }
    }
}

```

```
        System.out.println("Test client list");
        System.out.println(Server.clientList);
        Server.executor.remove(this);
        break;
    }
    System.out.println("Server received '" + input + "' from Client " + name);
    handleSendData(Server.clientList, input);
}
System.out.println("Closed socket for client " + socket.toString());
in.close();
out.close();
socket.close();

System.out.println(Server.executor.remove(this));
System.out.println(threadSet);
} catch (IOException e) {
    System.out.println(e);
}
}
}
```

Bài 5:

Viết chương trình chat giữa **nhiều clients** thông qua server, theo mô hình client/server, sử dụng giao thức TCP và dùng cơ chế multithread:

- Main thread trên server mở sẵn port 1235 và chờ các clients.
- Mỗi khi 1 client kết nối, server tạo 1 thread tương ứng để phục vụ client.
- Mỗi client gửi tin nhắn **chat 1-1** theo cú pháp **id#Nội dung tin nhắn** với **id** là số thứ tự của thread mà client muốn gửi tin nhắn tới (có thể xem như là tên của client, SV tự tùy chỉnh lại nếu muốn). Trong trường hợp client **id** chưa kết nối, server sẽ trả về thông báo lỗi cho client đã gửi tin.
- Ngoài ra, client có thể **chat 1 - all** theo cú pháp **all#Nội dung tin nhắn** với all là từ khóa cho biết tin nhắn sẽ gửi broadcast. Trong trường hợp này, chỉ những client đang kết nối mới nhận được tin nhắn.
- Khi có 1 client kết thúc bằng cách gửi chuỗi **bye**, server sẽ gửi thông báo kết thúc chat đến tất cả client còn lại và kết thúc thread của client đó.
- Main thread sẽ chỉ kết thúc khi người dùng chủ động dừng chương trình (hoặc sau 10 phút mà không có bất kỳ client nào kết nối – SV tự cài đặt)

```
public class Client {  
  
    private static String host = "localhost";  
    private static int port = 1234;  
    private static Socket socket;  
  
    public static void main(String[] args) throws IOException {  
        try {  
            socket = new Socket(host, port);  
            System.out.println("Client connected");  
            BufferedWriter out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));  
            BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
  
            Send send = new Send(socket, out, "bye");  
            Receive recv = new Receive(socket, in);  
            ExecutorService excutor = Executors.newFixedThreadPool(2);  
            excutor.execute(send);  
            excutor.execute(recv);  
  
        } catch (IOException e) {  
            System.out.println(e);  
        }  
    }  
}
```

```

public class ClientHandler implements Runnable {
    private Socket socket;
    private String name;
    BufferedReader in;
    BufferedWriter out;

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

    public void handleSendDataID(Vector<ClientHandler> clientList, String inputId, String inputData) throws IOException {
        int error = 0;
        for (ClientHandler client : clientList) {
            //Nếu người dùng nhập exit
            if (inputId.equals("") && inputData.equals("")) {
                if (!name.equals(client.name)) {
                    client.out.write("User " + name + " has been disconnected \n");
                    client.out.flush();
                }
            }
            //Nếu người dùng bắt đầu với cú pháp all
            if (inputId.equals("all")) {
                if (!name.equals(client.name)) {
                    client.out.write(inputData + "\n");
                    client.out.flush();
                }
            } else {
                //Nếu người dùng bắt đầu với cú pháp có id
                if (client.name.equals(inputId)) {

                    client.out.write(inputData + "\n");
                    client.out.flush();
                    System.out.println("Server sent " + inputData + " from Client " + name + "--> Client " + inputId);
                } else {
                    if (name.equals(client.name)) {
                        //Nếu người dùng nhập id không tồn tại
                        if (Integer.parseInt(inputId) > clientList.size()) {
                            client.out.write("User doesn't exist \n");
                            client.out.flush();
                        }
                    }
                }
            }
        }
    }

    public void handleSendData(Vector<ClientHandler> clientList, String inputData) throws IOException {
        for (ClientHandler client : clientList) {
            if (!name.equals(client.name)) {
                client.out.write(inputData + "\n");
                client.out.flush();
            }
        }
    }

    public void run() {
        System.out.println("Client " + socket.toString() + " accepted");
        Set<Thread> threadSet = Thread.getAllStackTraces().keySet();
        System.out.println(threadSet);
        try {
            String inputId = null;
            String inputData = null;
            while (true) {
                String input = in.readLine();
                //Handle if user's input is bye
                if (input.equals("bye")) {
                    handleSendDataID(Server.clientList, "", "");
                    handleSendData(Server.clientList,"");
                    System.out.println("Name " + name);
                    Server.clientList.remove(this);

                    System.out.println("Test client list");
                    System.out.println(Server.clientList);
                    Server.executor.remove(this);
                    break;
                }
            }
        }
    }
}

```

```

        if (input.contains("#")) {
            String[] listInputs = input.split("#");
            inputId = listInputs[0];
            inputData = listInputs[1];
        } else {
            break;
        }
        System.out.println("Server received '" + input + "' from Client " + name);
        handleSendDataID(Server.clientList, inputId, inputData);
        handleSendData(Server.clientList, input);
    }
    System.out.println("Closed socket for client " + socket.toString());
    in.close();
    out.close();
    socket.close();
    System.out.println(Server.executor.remove(this));
    System.out.println(threadSet);
} catch (IOException e) {
    System.out.println(e);
}
}

public class Server {
    public static int port = 1234;
    public static int numThread = 10;
    public static Vector<ClientHandler> clientList = new Vector<>();
    public static ThreadPoolExecutor executor = (ThreadPoolExecutor) Executors.newFixedThreadPool(numThread);
    public static ScheduledExecutorService schedule = Executors.newScheduledThreadPool(1);
    public static CheckTime ct = new CheckTime();
    public static ServerSocket server;

    public static void main(String[] args) throws IOException {
        try {
            server = new ServerSocket(port);
            System.out.println("Server binding at port " + port);
            System.out.println("Waiting for client...");
            int i = 1;
            ScheduledFuture<?> scheduledFuture = schedule.scheduleAtFixedRate(ct, 50, 1, TimeUnit.SECONDS);
            if (!schedule.awaitTermination(50, TimeUnit.SECONDS)) {
                System.out.println("Shut down");
                scheduledFuture.cancel(true);
                schedule.shutdown();
                executor.shutdown();
            } else {
                while (true) {
                    Socket socket = server.accept();
                    ClientHandler client = new ClientHandler(socket, Integer.toString(i++));
                    clientList.add(client);
                    executor.execute(client);
                    scheduledFuture.cancel(true);
                }
            } catch (IOException e) {
                System.out.println(e);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        finally {
            if (server != null) {
                server.close();
            }
        }
    }
}

class CheckTime implements Runnable {
    @Override
    public void run() {
        System.out.println("stop time. Time: " + Instant.now());
        System.exit(0);
    }
}
}

```