Serveur multithread :

Client

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

class PositionClient {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        BufferedReader br = null; // pour lire du texte sur la socket
        PrintStream ps = null; // pour écrire du texte sur la socket
        int clientId = -1;

        Socket sock = null;
        int port = -1;

        if (args.length != 2) {
            System.out.println("usage: PositionClient ip_server port");
            System.exit(1);
        }

        try {
            port = Integer.parseInt(args[1]);
            sock = new Socket(args[0], port);
        } catch (IOException e) {
            System.out.println("problème de connexion au serveur : " +
e.getMessage());
            System.exit(1);
        }

        try {
            br = new BufferedReader(new
InputStreamReader(sock.getInputStream()));
            ps = new PrintStream(sock.getOutputStream());

            System.out.println("Connexion au serveur en cours...");
            clientId = Integer.parseInt(br.readLine()); // première ligne
envoyée par le serveur = ID
            System.out.println("Connecté au serveur avec l'id: " +
clientId);

            while (true) {
                System.out.print("> ");
                if (!sc.hasNextLine()) {
                    break; // EOF ou CTRL+D
                }
                String input = sc.nextLine().trim();
```

```
                if (input.isEmpty())
                    continue;

                // Quitter proprement
                if (input.equalsIgnoreCase("exit")) {
                    ps.println("exit");
                    String resp = br.readLine();
                    System.out.println("Server: " + resp);
                    break;
                }

                if (input.startsWith("storepos ")) {
                    // Format attendu : storepos x,y,z
                    String coords =
input.substring("storepos".length()).trim();
                    if (!coords.matches("^-?\\d+(\\.\\d+)?,-?\\d+
(\\.\\d+)?,-?\\d+(\\.\\d+)?$")) {
                        System.out.println("Format invalide. Utilise :
storepos x,y,z");
                        continue;
                    }

                    // Envoi au serveur
                    ps.println("1"); // numéro de requête
                    ps.println(clientId); // ID
                    ps.println(coords); // coordonnées

                } else if (input.equals("pathlen")) {
                    // Envoi au serveur
                    ps.println("2"); // numéro de requête
                    ps.println(clientId); // ID

                } else if (input.startsWith("findpos ")) {
                    // Format attendu : findpos p,x,y,z
                    String params =
input.substring("findpos".length()).trim();
                    if (!params.matches("^-?\\d+(\\.\\d+)?,-?\\d+
(\\.\\d+)?,-?\\d+(\\.\\d+)?,-?\\d+(\\.\\d+)?$")) {
                        System.out.println("Format invalide. Utilise :
findpos p,x,y,z");
                        continue;
                    }

                    // Envoi au serveur
                    ps.println("3"); // numéro de requête
                    ps.println(clientId); // ID
                    ps.println(params); // facteur + coordonnées

                } else if (input.equals("pathelev")) {
                    // Envoi au serveur
                    ps.println("4"); // numéro de requête
                    ps.println(clientId); // ID

                } else if (input.equals("lastpos")) {
```

```
                    // Envoi au serveur
                    ps.println("5"); // numéro de requête
                    ps.println(clientId); // ID

                } else {
                    System.out.println("Commande inconnue. Commandes
valides :");
                    System.out.println("  storepos x,y,z");
                    System.out.println("  pathlen");
                    System.out.println("  findpos p,x,y,z");
                    System.out.println("  pathelev");
                    System.out.println("  lastpos");
                    System.out.println("  exit");
                    continue;
                }

                // Lecture de la réponse
                String resp = br.readLine();
                if (resp == null) {
                    System.out.println("Connexion fermée par le serveur.");
                    break;
                }

                switch (resp) {
                    case "REQ_ERR":
                        System.out.println("Erreur: format de requête
invalide.");
                        break;
                    case "NOID_ERR":
                        System.out.println("Erreur: ID client invalide.");
                        break;
                    default:
                        System.out.println("Server: " + resp);
                        break;
                }
            }

            br.close();
            ps.close();
            sock.close();

        } catch (IOException e) {
            System.out.println("Erreur: " + e.getMessage());
        }

        sc.close();
    }
}
```

Serveur

```
import java.io.*;
import java.net.*;
import java.util.ArrayList;
import java.util.HashMap;

class PositionServeur {
    private static int globalId = 0;
    private static final HashMap<Integer, ArrayList<Position>>
clientPositions = new HashMap<>();

    public static void main(String[] args) {
        if (args.length != 1) {
            System.out.println("usage: PositionServer port");
            System.exit(1);
        }

        int port = Integer.parseInt(args[0]);

        try (ServerSocket serverSocket = new ServerSocket(port)) {
            System.out.println("Serveur démarré sur le port " + port);

            while (true) {
                Socket clientSocket = serverSocket.accept(); // on accepte
un client
                synchronized (PositionServeur.class) {
                    clientPositions.put(globalId, new ArrayList<>());
                }

                // Lancer un nouveau thread pour ce client
                Thread t = new Thread(new ClientHandler(clientSocket,
globalId, clientPositions));
                t.start();

                globalId++;
            }
        } catch (IOException e) {
            System.out.println("Erreur serveur: " + e.getMessage());
        }
    }
}
```

Client handler

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

class ClientHandler implements Runnable {
    private final Socket socket;
    private final int clientId;
    private final HashMap<Integer, ArrayList<Position>> clientPositions;
```

```
    public ClientHandler(Socket socket, int clientId, HashMap<Integer,
ArrayList<Position>> clientPositions) {
        this.socket = socket;
        this.clientId = clientId;
        this.clientPositions = clientPositions;
    }

    @Override
    public void run() {
        try (BufferedReader br = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
                PrintStream ps = new PrintStream(socket.getOutputStream()))
{

            System.out.println("Client connecté. ID: " + clientId);
            ps.println(clientId); // on envoie l'ID attribué

            String input;
            while ((input = br.readLine()) != null) {
                if (input.equals("exit")) {
                    System.out.println("Client " + clientId + "
déconnecté.");
                    break;
                }

                int reqNumber;
                try {
                    reqNumber = Integer.parseInt(input);
                } catch (NumberFormatException e) {
                    ps.println("REQ_ERR");
                    continue;
                }

                // lecture de l'ID client
                input = br.readLine();
                int id;
                try {
                    id = Integer.parseInt(input);
                } catch (Exception e) {
                    ps.println("NOID_ERR");
                    continue;
                }

                if (!clientPositions.containsKey(id)) {
                    ps.println("NOID_ERR");
                    continue;
                }

                String output = "REQ_ERR"; // réponse par défaut
                switch (reqNumber) {
                    case 1 -> { // storepos
                        input = br.readLine();
                        String[] parts = input.split(",");
```

```
                        if (parts.length == 3) {
                            try {
                                double x = Double.parseDouble(parts[0]);
                                double y = Double.parseDouble(parts[1]);
                                double z = Double.parseDouble(parts[2]);
                                synchronized (clientPositions) {
                                    clientPositions.get(id).add(new
Position(x, y, z));
                                }
                                output = "OK";
                            } catch (NumberFormatException ignored) {
                            }
                        }
                    }
                    case 2 -> { // pathlen
                        double sum = 0.0;
                        ArrayList<Position> positions;
                        synchronized (clientPositions) {
                            positions = clientPositions.get(id);
                        }
                        if (positions.size() > 1) {
                            for (int i = 0; i < positions.size() - 1; i++)
{
                                sum +=
positions.get(i).distanceTo(positions.get(i + 1));
                            }
                        }
                        output = String.valueOf(sum);
                    }
                    case 3 -> { // findpos
                        input = br.readLine();
                        String[] parts = input.split(",");
                        if (parts.length == 4) {
                            try {
                                double prox = Double.parseDouble(parts[0]);
                                double x = Double.parseDouble(parts[1]);
                                double y = Double.parseDouble(parts[2]);
                                double z = Double.parseDouble(parts[3]);
                                Position p = new Position(x, y, z);

                                output = "FALSE";
                                synchronized (clientPositions) {
                                    for (Position pos :
clientPositions.get(id)) {
                                        if (pos.distanceTo(p) <= prox) {
                                            output = "TRUE";
                                            break;
                                        }
                                    }
                                }
                            } catch (NumberFormatException ignored) {
                            }
                        }
                    }
```

```
                    case 4 -> { // dénivelé
                        double denivpos = 0.0;
                        double denivneg = 0.0;
                        ArrayList<Position> positions;
                        synchronized (clientPositions) {
                            positions = clientPositions.get(id);
                        }
                        if (positions.size() > 1) {
                            for (int i = 0; i < positions.size() - 1; i++)
{
                                double dist = positions.get(i).getY() -
positions.get(i + 1).getY();
                                if (dist > 0)
                                    denivpos += dist;
                                else
                                    denivneg += dist;
                            }
                        }
                        output = "Denivele + : " + denivpos + ", Denivele -
: " + denivneg;
                    }
                    case 5 -> { // dernières positions des autres clients
                        StringBuilder response = new StringBuilder();
                        synchronized (clientPositions) {
                            for (Integer otherId :
clientPositions.keySet()) {
                                if (otherId != id &&
!clientPositions.get(otherId).isEmpty()) {
                                    Position lastPos =
clientPositions.get(otherId)

.get(clientPositions.get(otherId).size() - 1);
                                    response.append("Client
").append(otherId)
                                            .append(":
").append(lastPos.toString()).append(", ");
                                }
                            }
                        }
                        output = response.isEmpty() ? "Aucune position
trouvée" : response.toString();
                    }
                    default -> {
                        // REQ_ERR reste par défaut
                    }
                }

                ps.println(output);
            }

        } catch (IOException e) {
            System.out.println("Erreur client " + clientId + " : " +
e.getMessage());
        }
```

```
        }
    }
```