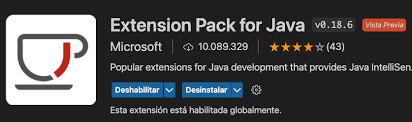
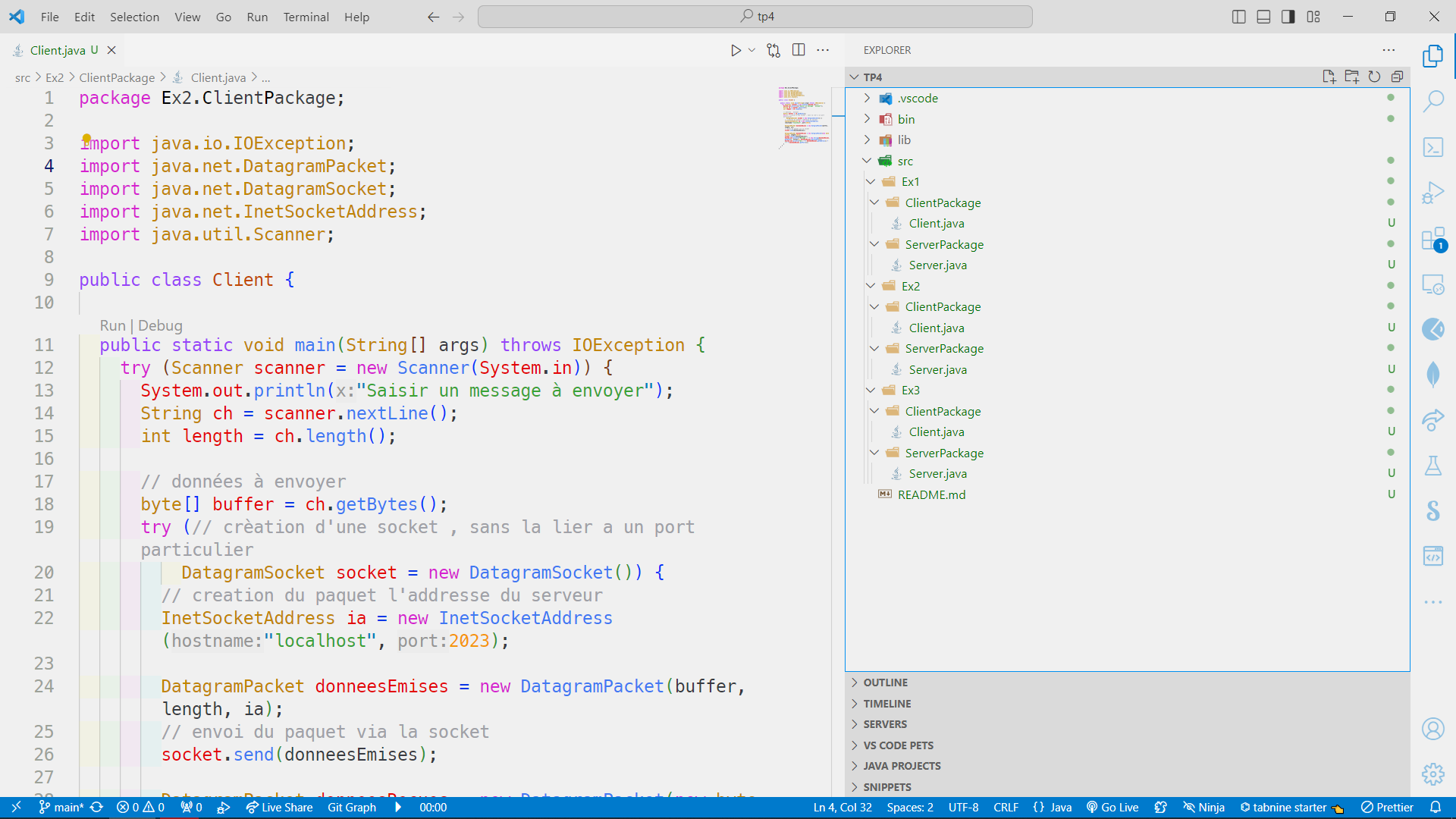
|  |
| --- |
| **Compte Rendu** |
| Tp1 : Développement d’applications réparties  Hammami Mouhamed Yassine  **GLSI3-1C-gp2** |

Environnement de travail :

* L’éditeur du text VSCode + Java Extension pack :

* Structure de Projet :



* Ex1

Server.java

**🡪Dans la boucle while on accepte et on lance les processus qui implémentent l’interface Runnable avec la méthode start() .**

* **Client.java**



**🡪Lancement 10 processus pour tester le multithreading.**

* **ClientProcess.java**

package Ex1.Methode1.ClientPackage;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.io.OutputStream;

import java.io.PrintWriter;

import java.net.Socket;

public class Client {

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

    for (int i = 0; i < 10; i++) {

      Socket socketClient = new Socket("localhost", 2023);

      InputStream cltInput = socketClient.getInputStream();

      InputStreamReader clientIsr = new InputStreamReader(cltInput);

      BufferedReader clientBfrIn = new BufferedReader(clientIsr);

      // Pour envoyer des flux de données au serveur

      OutputStream cltOutput = socketClient.getOutputStream();

      //

      PrintWriter pw = new PrintWriter(cltOutput, true);

      //

      pw.println("Eco pour Client " + (i + 1));

      System.out.println("client " + (i + 1) + " a envoye " + (9999 + i));

      System.out.println("response server pour client " + (i + 1) + "= " + clientBfrIn.readLine());

      socketClient.close();

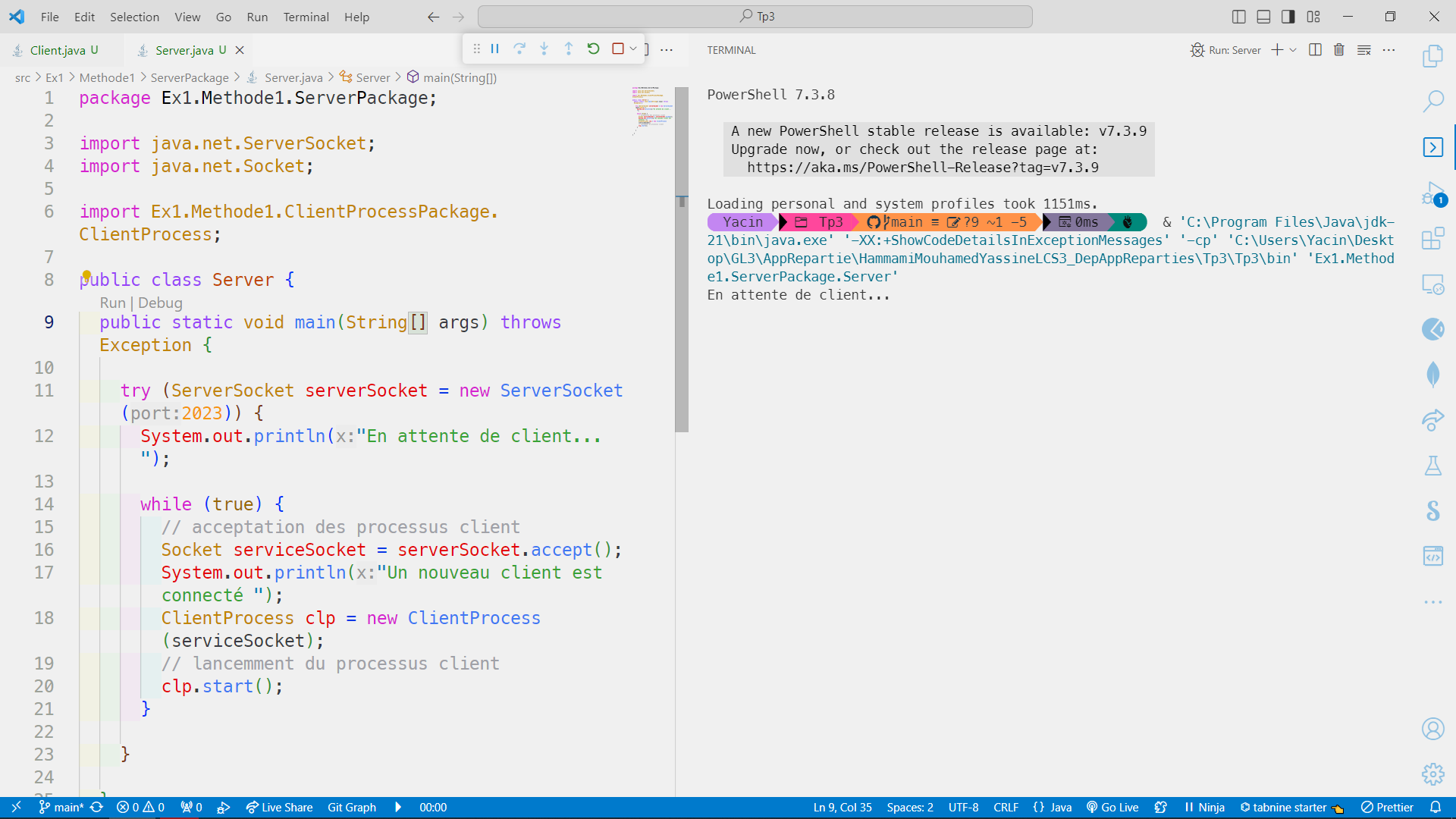
    }

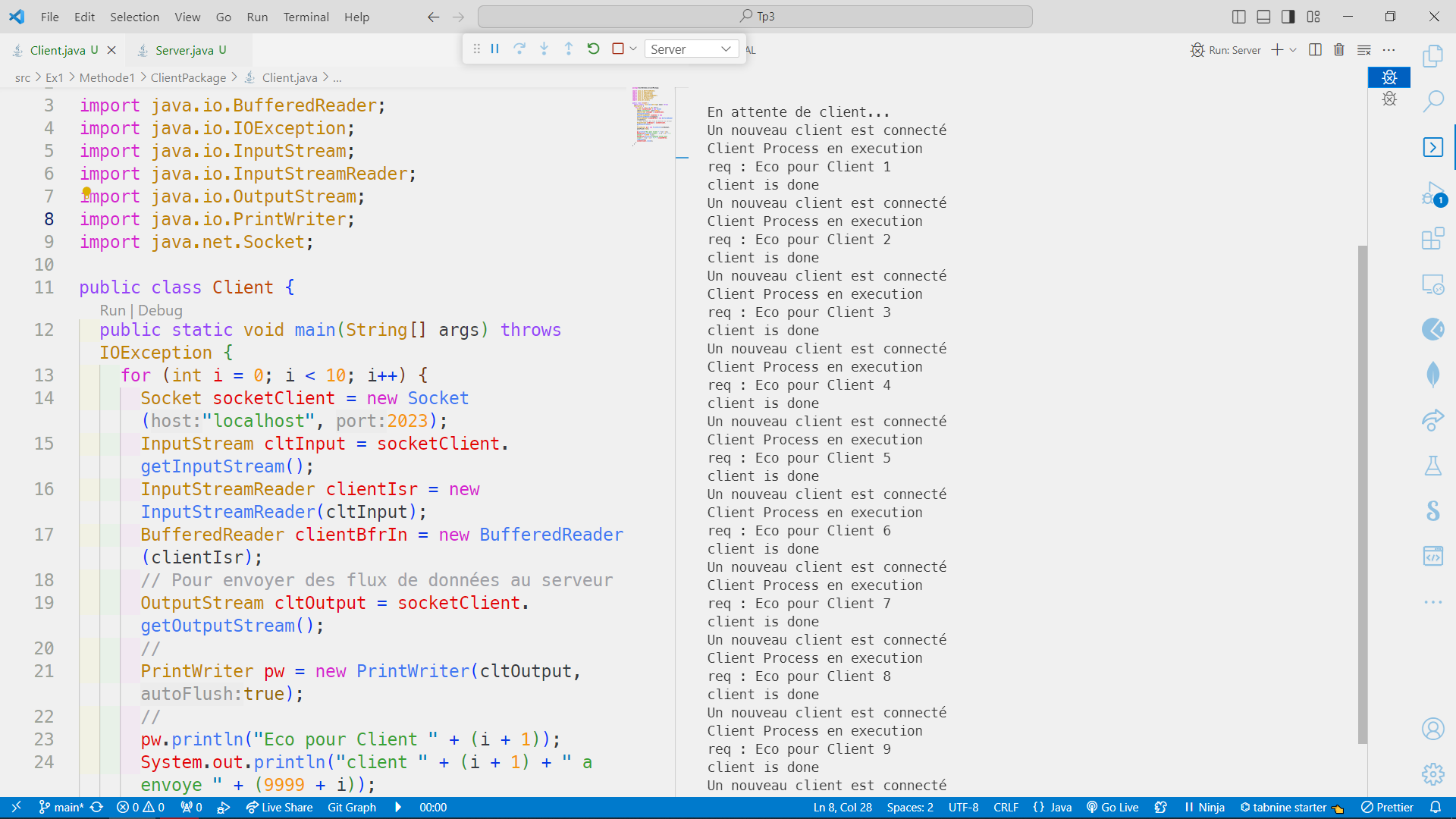
  }

}

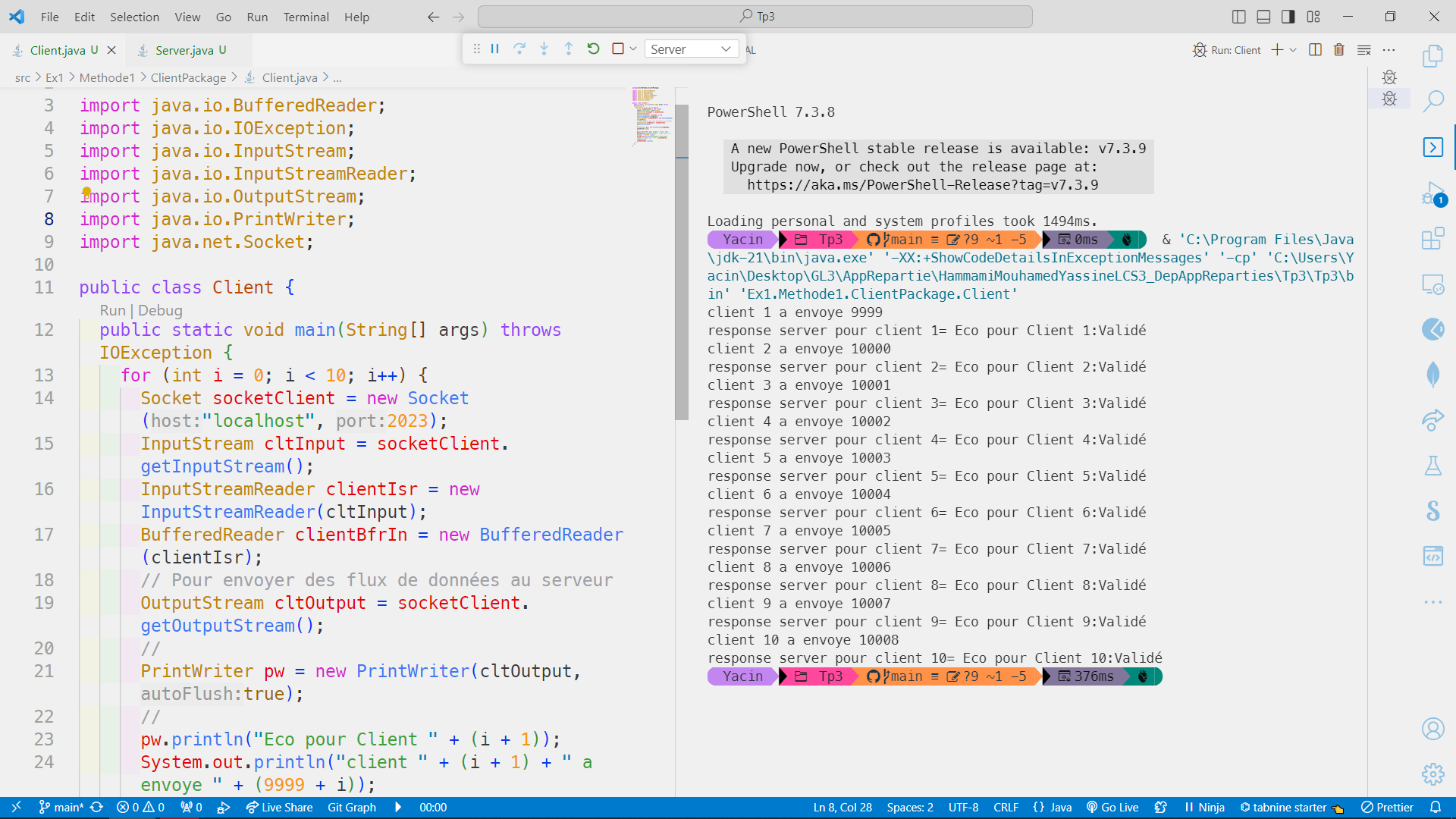
***Exécution :***

***Server***





***Client :***



***Méthode 2 (seulement deux classes) :***

***NB : Il n’y a pas de changement dans la class Client.java***

***Server.java :***

package Ex1.Methode2.ServerPackage;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.io.OutputStream;

import java.io.PrintWriter;

import java.net.ServerSocket;

import java.net.Socket;

public class Server extends Thread {

  public void run() {

    try (ServerSocket serverSocket = new ServerSocket(2023)) {

      System.out.println("En attente de client...");

      while (true) {

        Socket serviceSocket = serverSocket.accept();

        System.out.println("Un nouveau client est connecté ");

        ClientProcess clp = new ClientProcess(serviceSocket);

        clp.start();

      }

    } catch (IOException e) {

      e.printStackTrace();

    }

  }

}

class ClientProcess extends Thread {

  Socket socket;

  public ClientProcess(Socket socket) {

    super();

    this.socket = socket;

  }

  @Override

  public void run() {

    System.out.println("Client Process en execution");

    try {

      InputStream clpInput = this.socket.getInputStream();

      InputStreamReader reader = new InputStreamReader(clpInput);

      BufferedReader clpBfr = new BufferedReader(reader);

      // Pour envoyer des flux de données au serveur

      OutputStream clpOutput = this.socket.getOutputStream();

      PrintWriter clpWriter = new PrintWriter(clpOutput, true);

      // int input = clpInput.read();

      String req = clpBfr.readLine();

      System.out.println("req: " + req);

      clpWriter.println(req + ":Validé");

      System.out.println("client is done");

      // System.out.println("input: " + input);

      // System.out.println("Resutat envoyé");

      socket.close();

    } catch (Exception e) {

      e.printStackTrace();

    }

  }

  // lancement du server qui implemente l'interface Runnable

  public static void main(String[] args) {

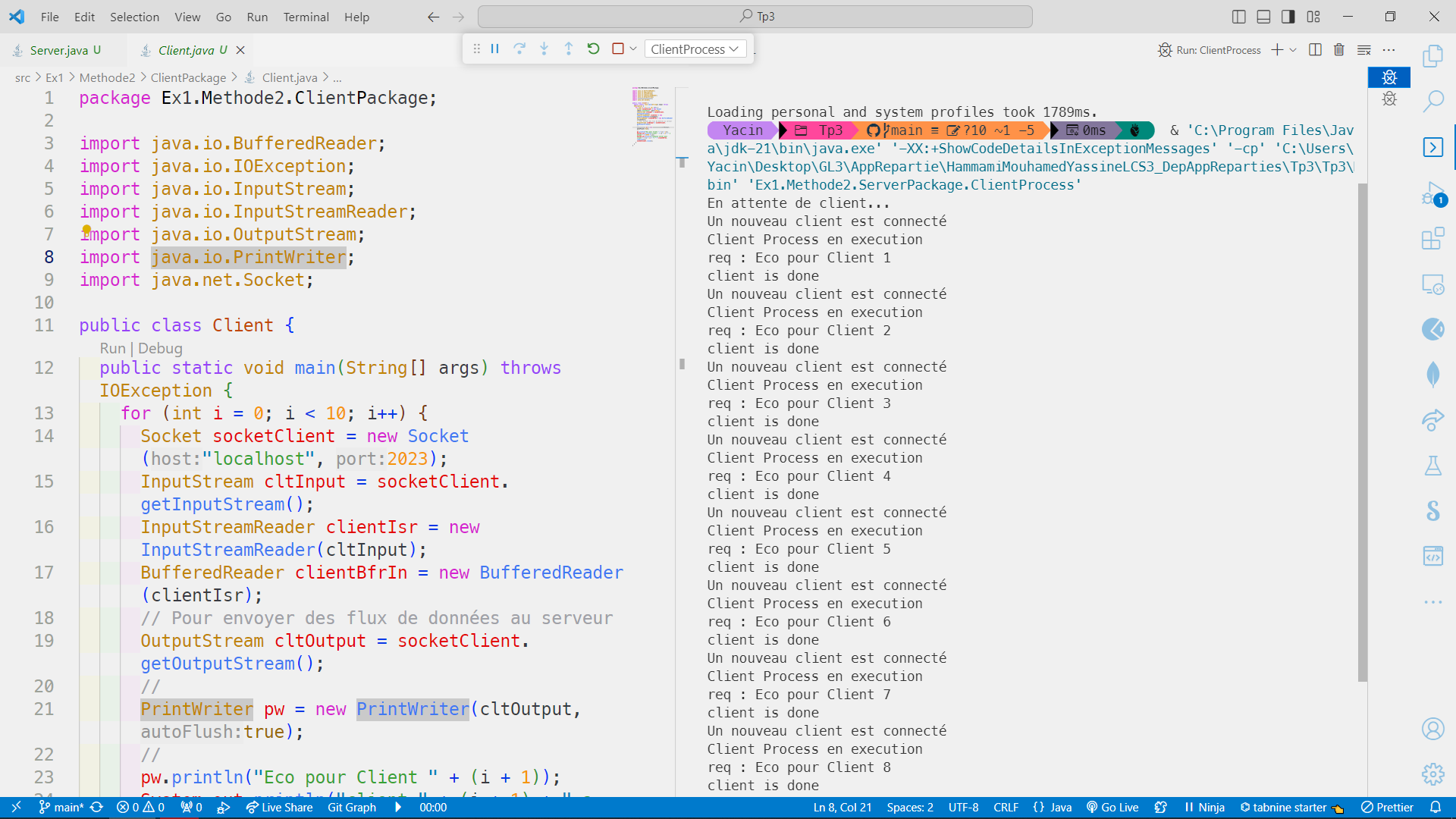
    new Server().start();

  }

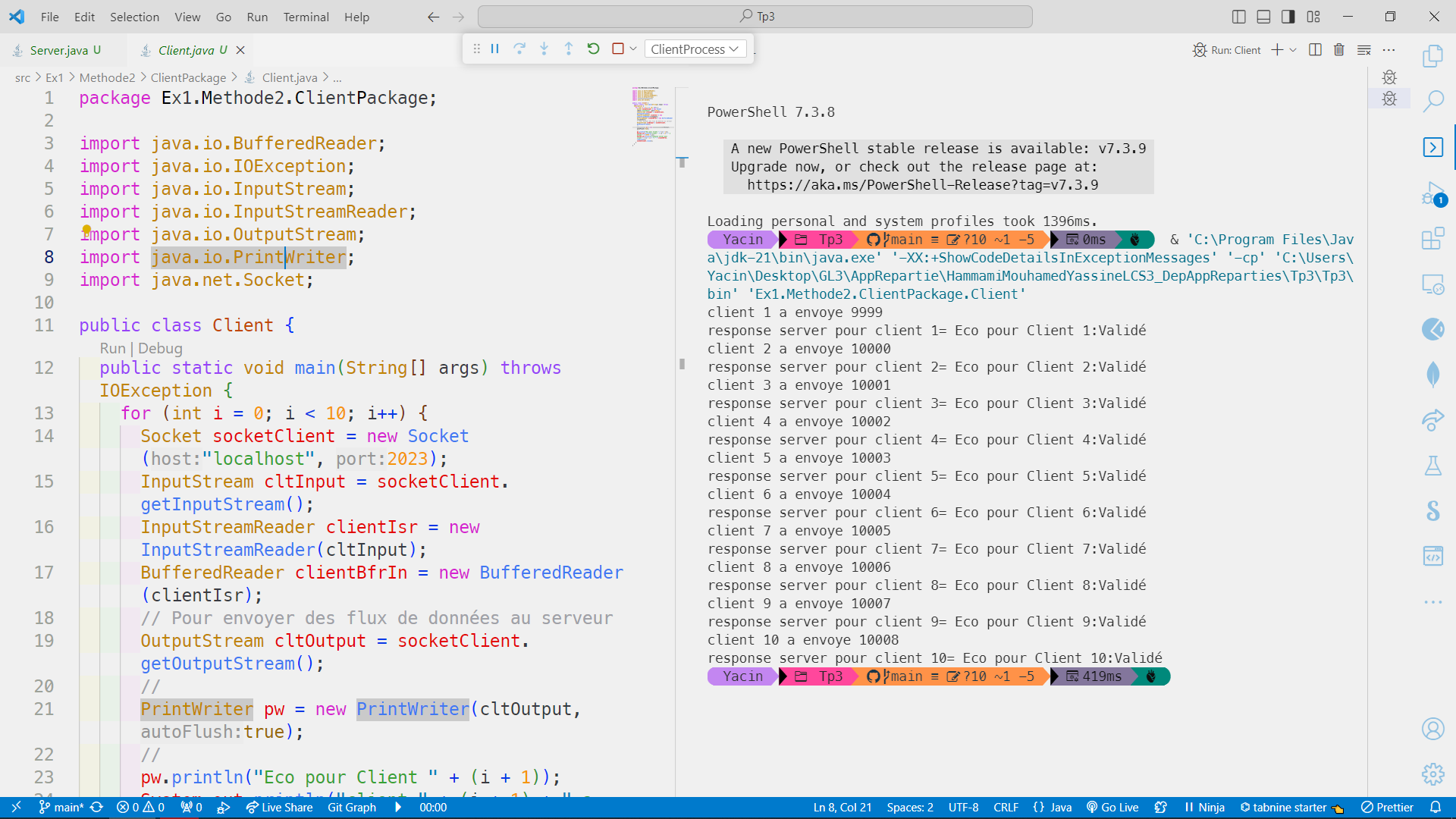
}

***Exécution :***

***Server :***



***Client :***



**Ex2 : Calculatrice :**

**Méthode 1 :**

**Server.java :**

package Ex2.Methode1.ServerPackage;

import java.net.ServerSocket;

import java.net.Socket;

import Ex2.Methode1.ClientProcessPackage.ClientProcess;

public class Server {

  public static void main(String[] args) throws Exception {

    try (ServerSocket serverSocket = new ServerSocket(2023)) {

      System.out.println("En attente de client...");

      while (true) {

        Socket serviceSocket = serverSocket.accept();

        System.out.println("Un nouveau client est connecté ");

        ClientProcess clp = new ClientProcess(serviceSocket);

        clp.start();

      }

    }

  }

}

**Client.java :**

package Ex2.Methode1.ClientPackage;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.io.OutputStream;

import java.io.PrintWriter;

import java.net.Socket;

import java.util.Scanner;

public class Client {

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

    Scanner userInput = new Scanner(System.in);

    for (int i = 0; i < 10; i++) {

      Socket socketClient = new Socket("localhost", 2023);

      InputStream cltInput = socketClient.getInputStream();

      InputStreamReader clientIsr = new InputStreamReader(cltInput);

      BufferedReader clientBfrIn = new BufferedReader(clientIsr);

      // Pour envoyer des flux de données au serveur

      OutputStream cltOutput = socketClient.getOutputStream();

      //

      PrintWriter pw = new PrintWriter(cltOutput, true);

      //

      // Demander d'utlisateur d'ecrire

      System.out.println("donnez une operation pour envoyer au serveur");

      String val = userInput.nextLine();

      // envoi d'expression

      pw.println(val);

      // lire le résultat

      String res = clientBfrIn.readLine();

      System.out.println(res);

      socketClient.close();

    }

    userInput.close();

  }

}

**ClientProcess.java :**

package Ex2.Methode1.ClientProcessPackage;

import java.io.BufferedReader;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.io.OutputStream;

import java.io.PrintWriter;

import java.net.Socket;

public class ClientProcess extends Thread {

  Socket socket;

  public ClientProcess(Socket socket) {

    super();

    this.socket = socket;

  }

  @Override

  public void run() {

    System.out.println("Client Process en execution");

    try {

      InputStream clpInput = this.socket.getInputStream();

      InputStreamReader reader = new InputStreamReader(clpInput);

      BufferedReader clpBfr = new BufferedReader(reader);

      // Pour envoyer des flux de données au serveur

      OutputStream clpOutput = this.socket.getOutputStream();

      PrintWriter clpWriter = new PrintWriter(clpOutput, true);

      float value = 0;

      String expression = clpBfr.readLine();

      try {

        // ==> On assume que l'oeration est de la forme suivante :3+4/3-4/3/4/3\*4 alors

        // si une opérateur existe dans l'expression alors son index est le maximum

        // parmis les autre opérateurs

        int pos = Math.max(expression.indexOf('+'),

            Math.max(expression.indexOf('-'), Math.max(expression.indexOf('/'), expression.indexOf('\*'))));

        System.out.println("pos=" + pos);

        char opr = expression.charAt(pos);

        System.out.println("opr=" + opr);

        // extraction des opérandes

        float opr1 = Float.parseFloat(expression.substring(0, pos));

        float opr2 = Float.parseFloat(expression.substring(pos + 1));

        // la logique des calcul

        switch (opr) {

          case '+':

            value = opr1 + opr2;

            break;

          case '-':

            value = opr1 - opr2;

            break;

          case '/':

            value = opr1 / opr2;

            break;

          case '\*':

            value = opr1 \* opr2;

            break;

          default:

            System.out.println("Opération ivalide");

            System.exit(1);

            break;

        }

        // gestion des exceptions

      } catch (Exception e) {

        clpWriter.println("Opération ivalide");

        System.out.println(e.toString());

        socket.close();

        Thread.currentThread().interrupt();

        // System.exit(1);

      }

      // envoi des résultas valide

      System.out.println("Result : " + value);

      clpWriter.println("Result : " + value);

      socket.close();

    } catch (Exception e) {

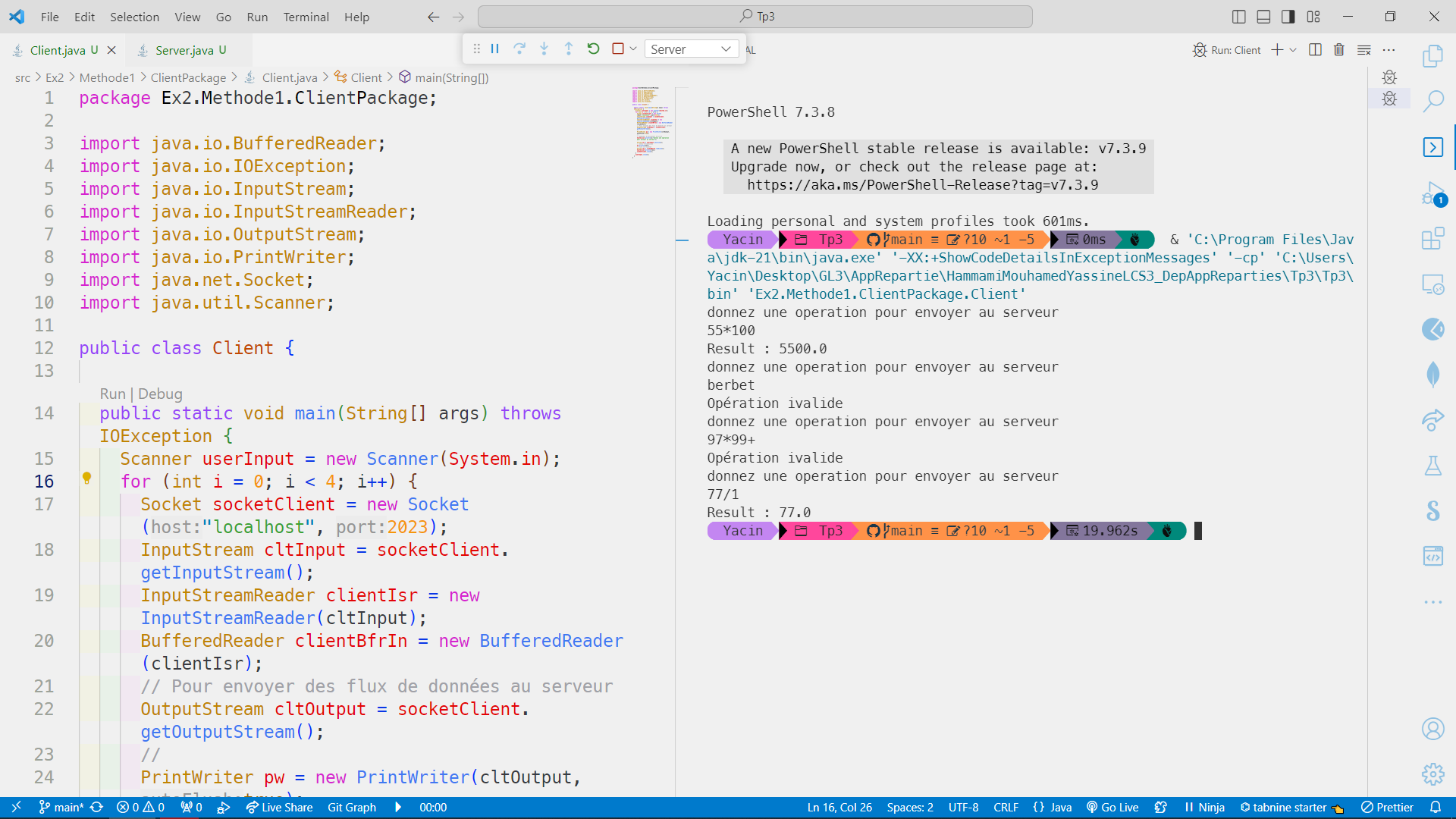
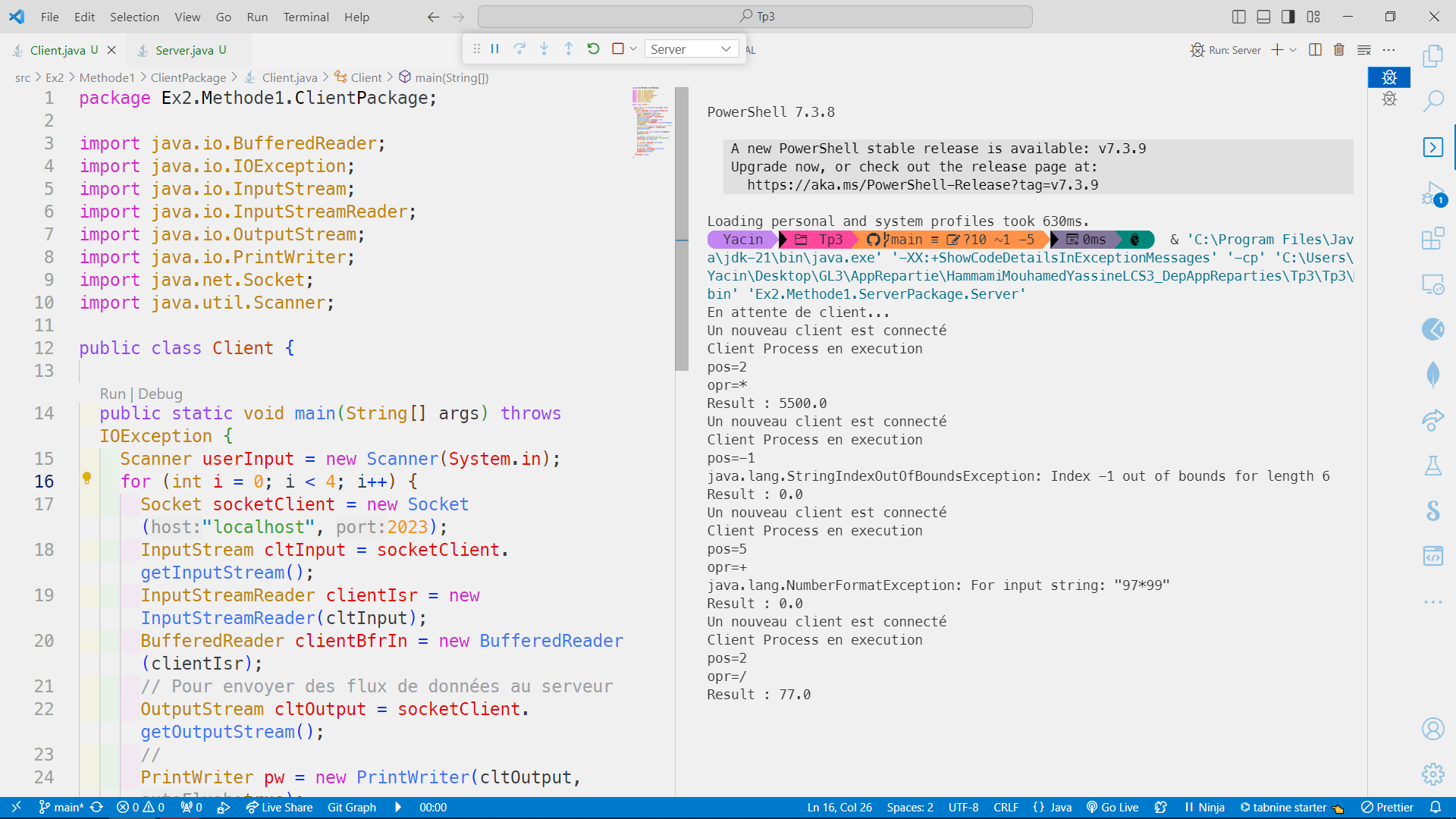
      e.printStackTrace();

    }

  }

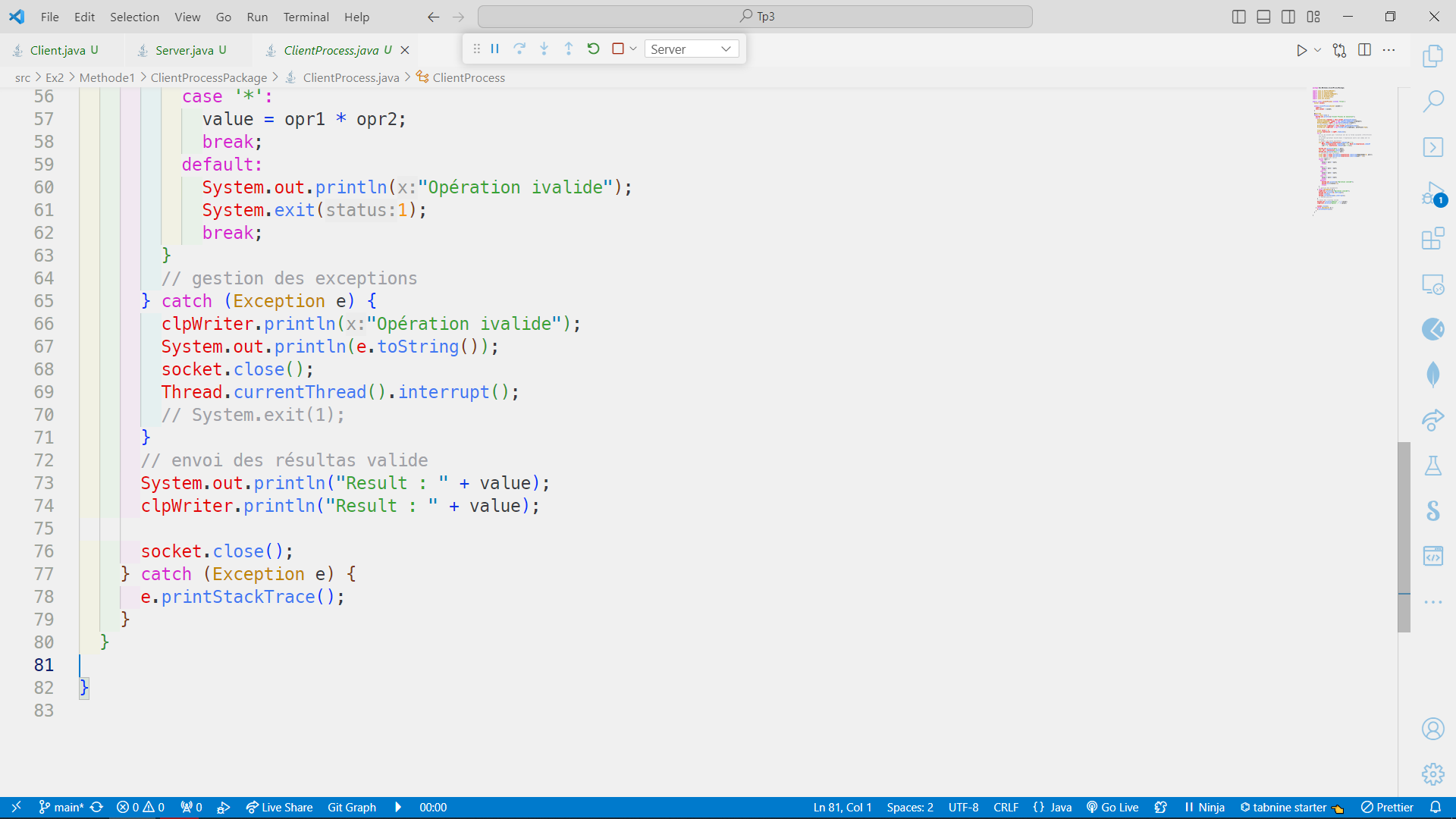
}

**Exécution :**



**Remarque :**

**La gestion des exception des opération invalide se fait avec la méthode Thread.interrupt() non par System.exit() .**



**Méthode 2 :**

**Client.java**

package Ex2.Methode2.ClientPackage;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.io.OutputStream;

import java.io.PrintWriter;

import java.net.Socket;

import java.util.Scanner;

public class Client {

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

    Scanner userInput = new Scanner(System.in);

    for (int i = 0; i < 10; i++) {

      Socket socketClient = new Socket("localhost", 2023);

      InputStream cltInput = socketClient.getInputStream();

      InputStreamReader clientIsr = new InputStreamReader(cltInput);

      BufferedReader clientBfrIn = new BufferedReader(clientIsr);

      // Pour envoyer des flux de données au serveur

      OutputStream cltOutput = socketClient.getOutputStream();

      //

      PrintWriter pw = new PrintWriter(cltOutput, true);

      //

      // Demander d'utlisateur d'ecrire

      System.out.println("donnez une operation pour envoyer au serveur");

      String val = userInput.nextLine();

      // envoi d'expression

      pw.println(val);

      // lire le résultat

      String res = clientBfrIn.readLine();

      System.out.println(res);

      socketClient.close();

    }

    userInput.close();

  }

}

**Server.java**

package Ex2.Methode2.ServerPackage;

import java.net.ServerSocket;

import java.net.Socket;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.io.OutputStream;

import java.io.PrintWriter;

public class Server extends Thread {

  public void run() {

    try (ServerSocket serverSocket = new ServerSocket(2023)) {

      System.out.println("En attente de client...");

      while (true) {

        Socket serviceSocket = serverSocket.accept();

        System.out.println("Un nouveau client est connecté ");

        ClientProcess clp = new ClientProcess(serviceSocket);

        clp.start();

      }

    } catch (IOException e) {

      e.printStackTrace();

    }

  }

  public class ClientProcess extends Thread {

    Socket socket;

    public ClientProcess(Socket socket) {

      super();

      this.socket = socket;

    }

    @Override

    public void run() {

      System.out.println("Client Process en execution");

      try {

        InputStream clpInput = this.socket.getInputStream();

        InputStreamReader reader = new InputStreamReader(clpInput);

        BufferedReader clpBfr = new BufferedReader(reader);

        // Pour envoyer des flux de données au serveur

        OutputStream clpOutput = this.socket.getOutputStream();

        PrintWriter clpWriter = new PrintWriter(clpOutput, true);

        float value = 0;

        String expression = clpBfr.readLine();

        try {

          // ==> On assume que l'oeration est de la forme suivante :3+4/3-4/3/4/3\*4 alors

          // si une opérateur existe dans l'expression alors son index est le maximum

          // parmis les autre opérateurs

          int pos = Math.max(expression.indexOf('+'),

              Math.max(expression.indexOf('-'), Math.max(expression.indexOf('/'), expression.indexOf('\*'))));

          System.out.println("pos=" + pos);

          char opr = expression.charAt(pos);

          System.out.println("opr=" + opr);

          // extraction des opérandes

          float opr1 = Float.parseFloat(expression.substring(0, pos));

          float opr2 = Float.parseFloat(expression.substring(pos + 1));

          // la logique des calcul

          switch (opr) {

            case '+':

              value = opr1 + opr2;

              break;

            case '-':

              value = opr1 - opr2;

              break;

            case '/':

              value = opr1 / opr2;

              break;

            case '\*':

              value = opr1 \* opr2;

              break;

            default:

              System.out.println("Opération ivalide");

              System.exit(1);

              break;

          }

          // gestion des exceptions

        } catch (Exception e) {

          clpWriter.println("Opération ivalide");

          System.out.println(e.toString());

          socket.close();

          Thread.currentThread().interrupt();

          // System.exit(1);

        }

        // envoi des résultas valide

        System.out.println("Result : " + value);

        clpWriter.println("Result : " + value);

        socket.close();

      } catch (Exception e) {

        e.printStackTrace();

      }

    }

  }

  public static void main(String[] args) {

    new Server().start();

  }

}

**Exécution :**

