

# Compte rendu

Activité pratique n° 2

## Bases des systèmes distribués – prog réseau



**HOUD Fatima-Ezzahra**  
[f.houd@etu.enset-media.ac.ma](mailto:f.houd@etu.enset-media.ac.ma)



# TABLE DES MATIÈRES

Introduction .....	2
Énoncé.....	3
Code source .....	4
Captures .....	14
Conclusion .....	18

# INTRODUCTION

Ce présent écrit est rédigé comme compte rendu pour les travaux pratiques effectués le 06 février 2023 dans le cadre du module « Architecture JEE et Middlewares » tout particulièrement les systèmes distribués -programmation réseau.

Les systèmes distribués sont des systèmes informatiques constitués de plusieurs ordinateurs interconnectés, qui travaillent ensemble pour fournir une fonctionnalité globale. La programmation réseau est une partie importante de la programmation des systèmes distribués car elle permet aux différents composants du système de communiquer entre eux à travers un réseau.

La programmation réseau implique l'utilisation de sockets pour établir des connexions entre des ordinateurs sur un réseau. Un socket est une interface de programmation qui permet à un programme d'envoyer et de recevoir des données à travers un réseau.

La programmation réseau peut être effectuée dans de nombreux langages de programmation, tels que Python, Java, C++, etc. Les bibliothèques de programmation telles que les bibliothèques de sockets standard de Python, de Java ou de C++ fournissent des fonctions pour la création de sockets et la communication entre ordinateurs.

Les protocoles de communication, tels que TCP/IP, UDP, etc., sont également importants dans la programmation réseau, car ils définissent les règles et les procédures pour la communication entre ordinateurs. La programmation réseau implique souvent la mise en œuvre de ces protocoles.

En résumé, la programmation réseau est une compétence importante pour la programmation de systèmes distribués, car elle permet aux différents composants du système de communiquer entre eux à travers un réseau. Les sockets et les protocoles de communication sont des éléments clés de la programmation réseau.

Reprendre les exemple des démo vidéos pour développer un application client serveur de Chat

## Partie I. Modèle Multi Threads Blocking IO (java.io)

- Développer un serveur de Multi Thread Blocking IO de ChatServer
- Tester le serveur avec un client Telnet
- Créer un client Java avec une interface graphique JavaFX
- Créer un client Python ou un autre langage quelconque

## Partie II. Modèle Single Thread avec Non Blocking IO (java.nio)

- Développer un serveur de Single Thread utilisant des entrées sorties non bloquantes
- Tester le serveur avec un client Telnet, un client java et un client d'un autre langage

Partie III. Utiliser un outil Comme JMeeter pour tester les performances des deux serveurs

## I- Modèle multi threads blocking IO

### Serveur de Multi Thread Blocking IO de ChatServer

```
package org.fatiza.blocking;

import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

public class MultiThreadBlockingChatServer extends Thread {
    private List<Conversation> conversations = new ArrayList<>();
    int clientsCount = 0;
    public static void main(String[] args) {
        new MultiThreadBlockingChatServer().start();
    }
    @Override
    public void run() {
        System.out.println("The server is started using port 2001");
        try {
            ServerSocket serverSocket = new ServerSocket(2001);
            while (true) {
                Socket socket = serverSocket.accept();
                ++clientsCount;
                Conversation conversation = new Conversation(socket, clientsCount);
                conversations.add(conversation);
                conversation.start();
            }
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
    class Conversation extends Thread {
        private int clientId;
        private Socket socket;
        public Conversation(Socket socket, int clientId) {
            this.socket = socket;
            this.clientId = clientId;
        }
        @Override
        public void run() {
            try {
```

```

InputStream is = socket.getInputStream();
InputStreamReader isr = new InputStreamReader(is);
BufferedReader br = new BufferedReader(isr);
OutputStream os = socket.getOutputStream();
PrintWriter pw = new PrintWriter(os, true);
System.out.println("New connection from Client n°" + clientId + " IP= " +
socket.getRemoteSocketAddress());
pw.println("\t Welcome you are the client n°:" + clientId);
String request;
String message = "";
List<Integer> ids = new ArrayList<>();
while ((request = br.readLine()) != null) {
    if (request.contains("=>")) {
        ids = new ArrayList<>();
        String[] items = request.split("=>");
        String clients = items[0];
        message = items[1];
        if (clients.contains(",")) {
            String[] idsListStr = clients.split(",");
            for (String id : idsListStr) {
                ids.add(Integer.parseInt(id));
            }
        } else {
            ids.add(Integer.parseInt(clients));
        }
    } else {
        message = request;
        ids = conversations.stream().map(c -> c.clientId).collect(Collectors.toList());
    }
    System.out.println("\t New Request => " + request + " from " +
socket.getRemoteSocketAddress());
    broadcastMessage(message, socket, ids);
}
} catch (Exception e) {
    throw new RuntimeException(e);
}
}

public void broadcastMessage(String message, Socket from, List<Integer> clientIds) {
    try {
        for (Conversation conversation : conversations) {
            Socket socket = conversation.socket;
            if ((socket != from) && clientIds.contains(conversation.clientId)) {
                OutputStream os = socket.getOutputStream();
                PrintWriter printWriter = new PrintWriter(os, true);
                printWriter.println(message);
            }
        }
    } catch (IOException e) {

```

```

        throw new RuntimeException(e);
    }
}
}
}

```

## Client Java

```

package org.fatiza;

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

public class ChatClient {
    public static void main(String[] args) throws IOException {
        String serverAddress = "localhost";
        int port = 2001;
        Socket socket = new Socket(serverAddress, port);
        InputStream is=socket.getInputStream();
        InputStreamReader isr=new InputStreamReader(is);
        BufferedReader input = new BufferedReader(isr);
        OutputStream os=socket.getOutputStream();
        OutputStreamWriter osr=new OutputStreamWriter(os);
        PrintWriter output = new PrintWriter(osr, true);
        BufferedReader consoleInput = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Connected!");
        while (true) {
            String message = consoleInput.readLine();
            output.println(message);
            if (message.equals("exit")) {
                break;
            }
            String response = input.readLine();
            System.out.println(response);
        }
        socket.close();
    }
}

```

## client Java avec interface JavaFx

Tout d'abord, Voici la structure de mon projet ClientFx :



### - La classe Client

```
- package com.clientfx;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;

public class Client {
    private Socket socket;
    private BufferedReader in;
    private PrintWriter out;
    public void connect(String host, int port) throws IOException, IOException {
        socket = new Socket(host, port);
        in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        out = new PrintWriter(socket.getOutputStream(), true);
    }
    public void disconnect() throws IOException {
        socket.close();
    }
}
```



```

    }
    public void send(String message) {
        out.println(message);
    }

    public String receive() throws IOException {
        return in.readLine();
    }
}

```

#### - La classe ClientController

```

package com.clientfx;

import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;

public class ClientController {
    @FXML private TextField messageField;
    @FXML private Button sendButton;
    @FXML private Button connectButton;
    @FXML private Button disconnectButton;
    @FXML private TextArea chatArea;

    private Client client;
    @FXML
    public void initialize() {
        client = new Client();
        sendButton.setOnAction(event -> sendMessage());
        connectButton.setOnAction(event -> connect());
        disconnectButton.setOnAction(event -> disconnect());
    }
    private void sendMessage() {
        String message = messageField.getText();
        client.send(message);
        messageField.clear();
    }
    private void connect() {

```

```

try {
    client.connect("localhost", 2001); // replace with your server's host and port
    chatArea.appendText("Connected to server\n");
} catch (IOException e) {
    chatArea.appendText("Failed to connect to server\n");
    e.printStackTrace();
}
}

private void disconnect() {
    try {
        client.disconnect();
        chatArea.appendText("Disconnected from server\n");
    } catch (IOException e) {
        chatArea.appendText("Failed to disconnect from server\n");
        e.printStackTrace();
    }
}
}

```

## - La classe ClientFX

```

- package com.clientfx;

import com.clientfx.ClientController;
import javafx.application.Application;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class ClientFX extends Application {
    @Override
    public void start(Stage stage) throws Exception {
        FXMLLoader loader = new FXMLLoader(getClass().getResource("ClientView.fxml"));
        Parent root = loader.load();
        ClientController controller = loader.getController();
        controller.initialize();

        stage.setScene(new Scene(root, 600, 400));
        stage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}

```

## Client python

```
import socket

def main():
    server_address = "localhost"
    port = 2001
    with socket.socket() as client_socket:
        client_socket.connect((server_address, port))
        input_stream = client_socket.makefile('r', encoding='utf-8')
        output_stream = client_socket.makefile('w', encoding='utf-8', newline='\r\n')
        console_input = input
        print("Connected!")
        while True:
            message = console_input(" -> ")
            output_stream.write(message + '\r\n')
            output_stream.flush()
            if message == "exit":
                break
            response = input_stream.readline().rstrip('\r\n')
            print("Received from server:", response)
        client_socket.close()

if __name__ == '__main__':
    main()
```

## 2- Modèle multi threads blocking IO

### Serveur

```
package org.fatiza.nonblocking;

import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.channels.SelectionKey;
import java.nio.channels.Selector;
import java.nio.channels.ServerSocketChannel;
import java.nio.channels.SocketChannel;
import java.util.Iterator;
import java.util.Set;

public class SingleThreadNonBlockingChatServer {

    public static void main(String[] args) throws Exception {
        // Initialisation du Selector et du ServerSocketChannel
        Selector selector = Selector.open();
        ServerSocketChannel serverSocketChannel = ServerSocketChannel.open();
        serverSocketChannel.bind(new InetSocketAddress("localhost", 2002));
        serverSocketChannel.configureBlocking(false);
        serverSocketChannel.register(selector, SelectionKey.OP_ACCEPT);

        // Boucle principale
        while (true) {
            selector.select();
            Set<SelectionKey> selectedKeys = selector.selectedKeys();
            Iterator<SelectionKey> keyIterator = selectedKeys.iterator();

            while (keyIterator.hasNext()) {
                SelectionKey key = keyIterator.next();

                if (key.isAcceptable()) {
                    handleAccept(key, selector);
                } else if (key.isReadable()) {
                    handleRead(key);
                }

                keyIterator.remove();
            }
        }
    }

    private static void handleAccept(SelectionKey key, Selector selector) throws Exception {
```

```
// Acceptation de la connexion et enregistrement du SocketChannel pour lecture
ServerSocketChannel serverSocketChannel = (ServerSocketChannel) key.channel();
SocketChannel socketChannel = serverSocketChannel.accept();
socketChannel.configureBlocking(false);
socketChannel.register(selector, SelectionKey.OP_READ);

System.out.println("New Connection Accepted");
}

private static void handleRead(SelectionKey key) throws Exception {
    // Lecture des données envoyées par le client
    SocketChannel socketChannel = (SocketChannel) key.channel();
    ByteBuffer buffer = ByteBuffer.allocate(1024);
    socketChannel.read(buffer);
    String request = new String(buffer.array()).trim();

    System.out.println("Received Message => " + request);

    // Si le client envoie "exit", fermer la connexion
    if (request.equals("exit")) {
        socketChannel.close();
        System.out.println("Connection Closed");
    }
}
}
```

## Client java

```
- package org.fatiza.nonblocking;

import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.channels.SocketChannel;
import java.util.Scanner;

public class Client {
    public static void main(String[] args) throws Exception {
        // Connexion au serveur
        SocketChannel socketChannel = SocketChannel.open(new InetSocketAddress("localhost",
2002));
        // Création du scanner pour la lecture de l'entrée utilisateur
        Scanner scanner = new Scanner(System.in);
        // Boucle principale
        while (true) {
            // Lecture de l'entrée utilisateur
            String request = scanner.nextLine();
```

```

// Envoi des données au serveur
ByteBuffer byteBuffer = ByteBuffer.wrap(request.getBytes());
socketChannel.write(byteBuffer);
// Lecture de la réponse du serveur
ByteBuffer responseBuffer = ByteBuffer.allocate(1024);
int bytesRead = socketChannel.read(responseBuffer);
// Affichage de la réponse si elle est non vide
if (bytesRead > 0) {
    String response = new String(responseBuffer.array(), 0, bytesRead).trim();
    System.out.println("Response => " + response);
}
}
}
}

```

## Client python

```

import socket

# Connexion au serveur
server_address = ('localhost', 2002)
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect(server_address)

# Boucle principale
while True:
    # Lecture de l'entrée utilisateur
    request = input()
    # Envoi des données au serveur
    client_socket.send(request.encode())
    # Lecture de la réponse du serveur
    response = client_socket.recv(1024).decode().strip()
    # Affichage de la réponse si elle est non vide
    if response:
        print("Response ==>", response)

```

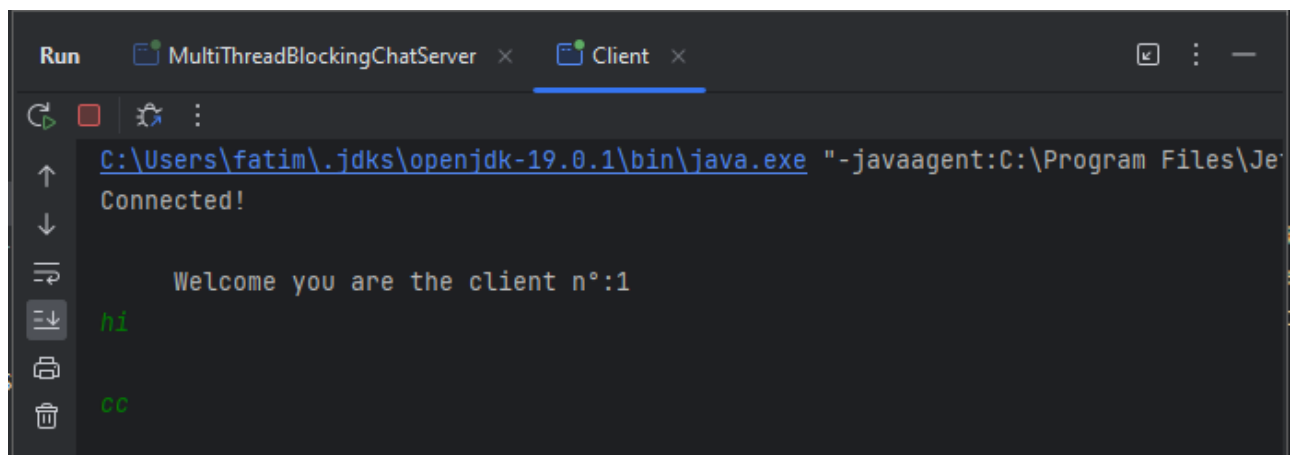
## I- Modèle multi threads blocking IO

### Serveur de Multi Thread Blocking IO de ChatServer



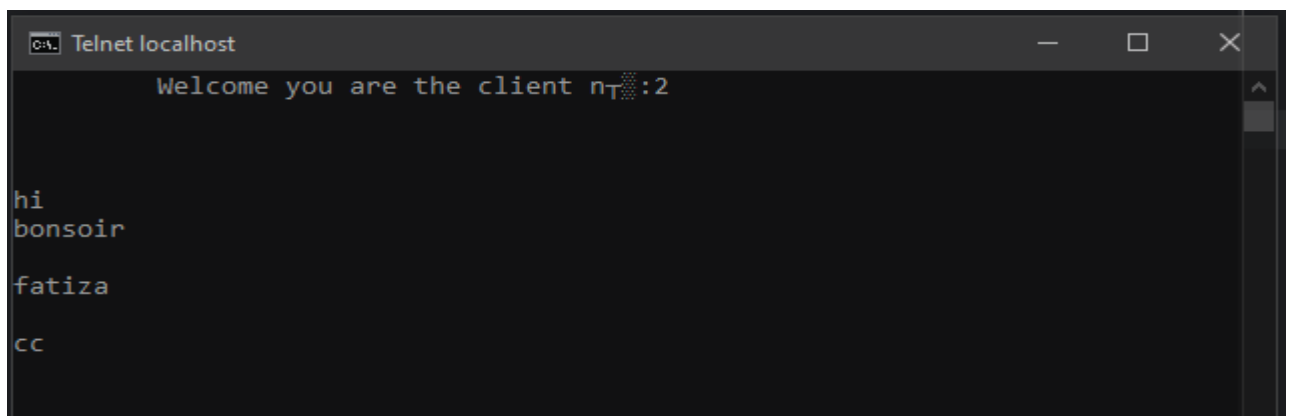
```
Run C:\Users\fatim\.jdk\openjdk-19.0.1\bin\java.exe "-javaagent:C:\Program
The server is started using port 2001
|
```

### Client Java



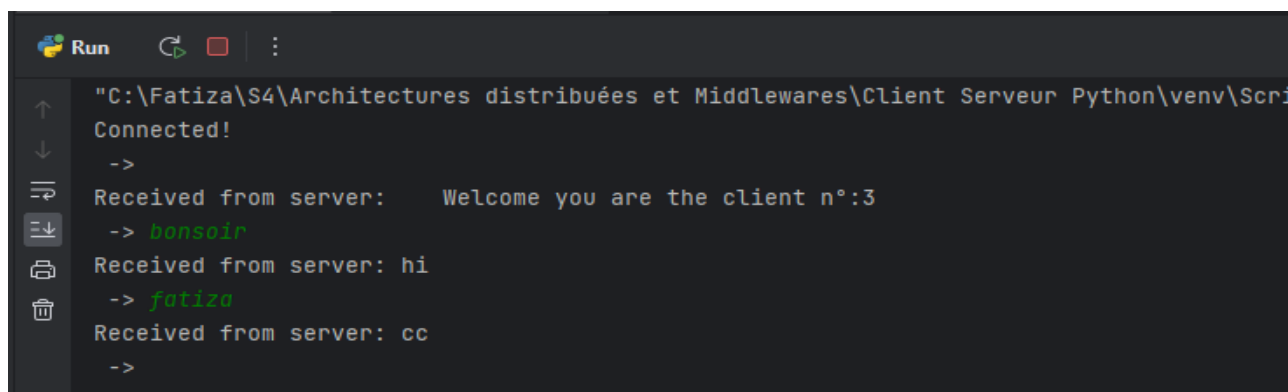
```
Run MultiThreadBlockingChatServer Client
C:\Users\fatim\.jdk\openjdk-19.0.1\bin\java.exe "-javaagent:C:\Program Files\Je
Connected!
Welcome you are the client n°:1
hi
cc
```

### Client Telnet



```
C:\> Telnet localhost
Welcome you are the client n°:2
hi
bonsoir
fatiza
cc
```

## Client python

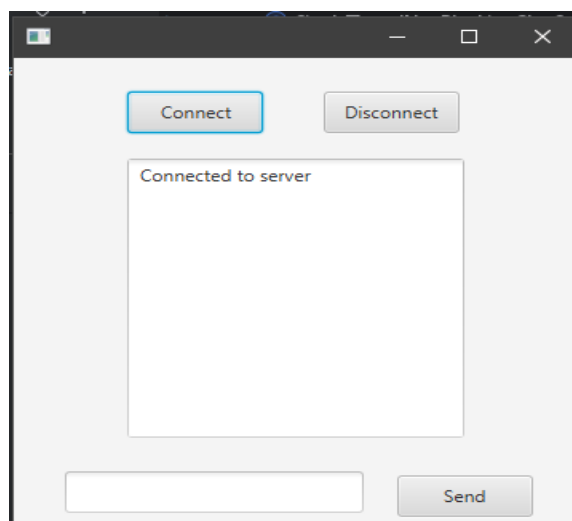
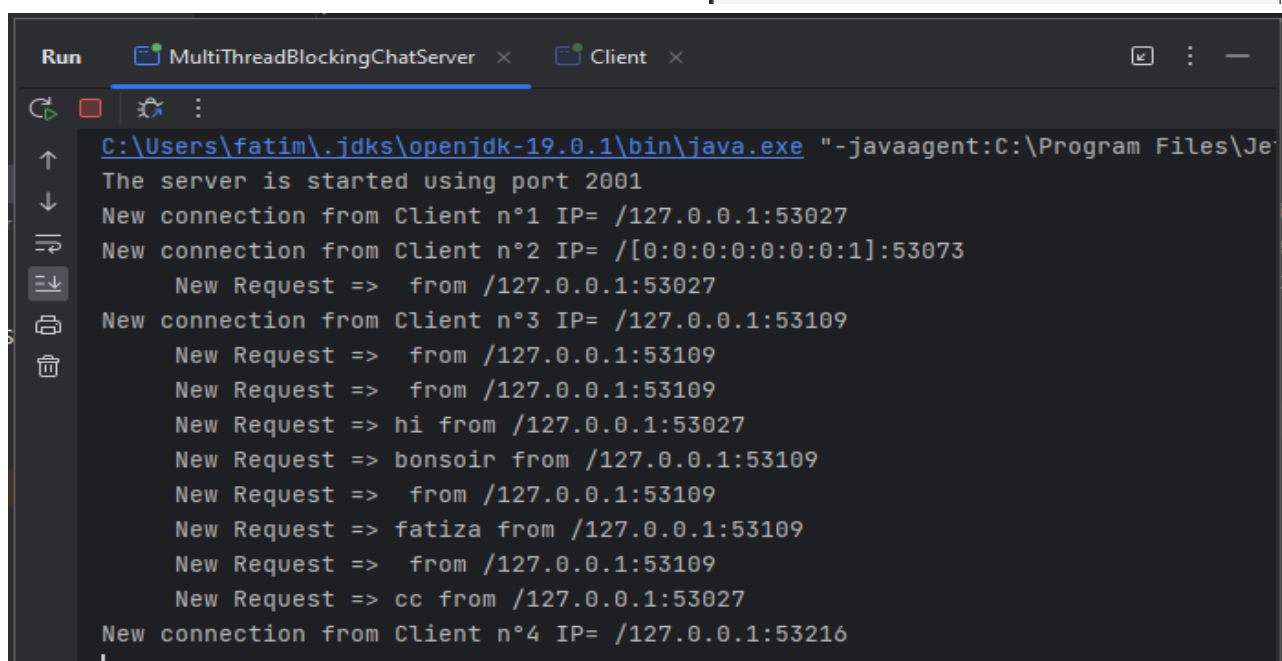


```

Run
"C:\Fatiza\S4\Architectures distribuées et Middlewares\Client Serveur Python\venv\Scripts\python.exe"
Connected!
->
Received from server: Welcome you are the client n°:3
-> bonsoir
Received from server: hi
-> fatiza
Received from server: cc
->

```

## Client JavaFX

```

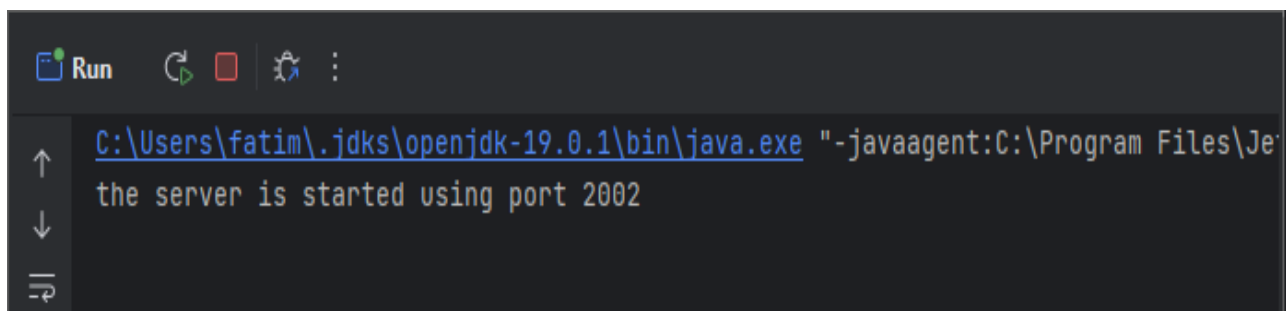
Run
MultiThreadBlockingChatServer x Client x
C:\Users\fatim\.jdk\openjdk-19.0.1\bin\java.exe "-javaagent:C:\Program Files\Java\jre-9\lib\jvmti.dll" -Djava.awt.headless=true -Djava.net.preferIPv4Stack=true -Djava.security.egd=file:/dev/./urandom -jar C:\Users\fatim\IdeaProjects\ChatServer\build\libs\chat-client.jar
The server is started using port 2001
New connection from Client n°1 IP= /127.0.0.1:53027
New connection from Client n°2 IP= /[0:0:0:0:0:0:1]:53073
New Request => from /127.0.0.1:53027
New connection from Client n°3 IP= /127.0.0.1:53109
New Request => from /127.0.0.1:53109
New Request => from /127.0.0.1:53109
New Request => hi from /127.0.0.1:53027
New Request => bonsoir from /127.0.0.1:53109
New Request => from /127.0.0.1:53109
New Request => fatiza from /127.0.0.1:53109
New Request => from /127.0.0.1:53109
New Request => cc from /127.0.0.1:53027
New connection from Client n°4 IP= /127.0.0.1:53216

```



## 2- Modèle multi threads blocking IO

### Serveur

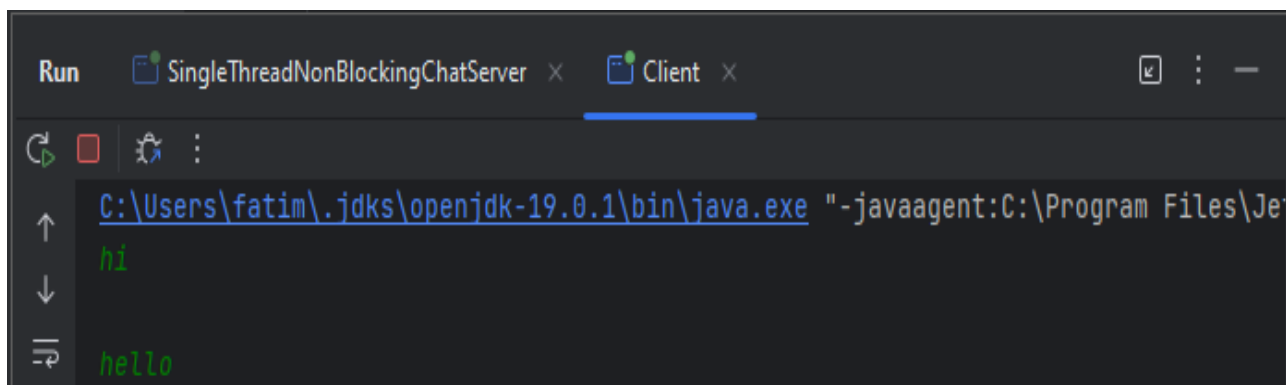


```

C:\Users\fatim\.jdk\openjdk-19.0.1\bin\java.exe "-javaagent:C:\Program Files\Je
the server is started using port 2002

```

### Client java

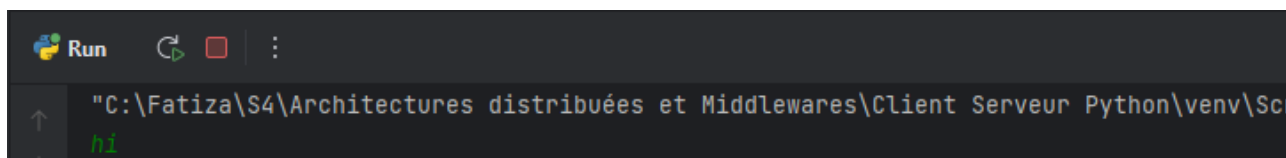


```

C:\Users\fatim\.jdk\openjdk-19.0.1\bin\java.exe "-javaagent:C:\Program Files\Je
hi
hello

```

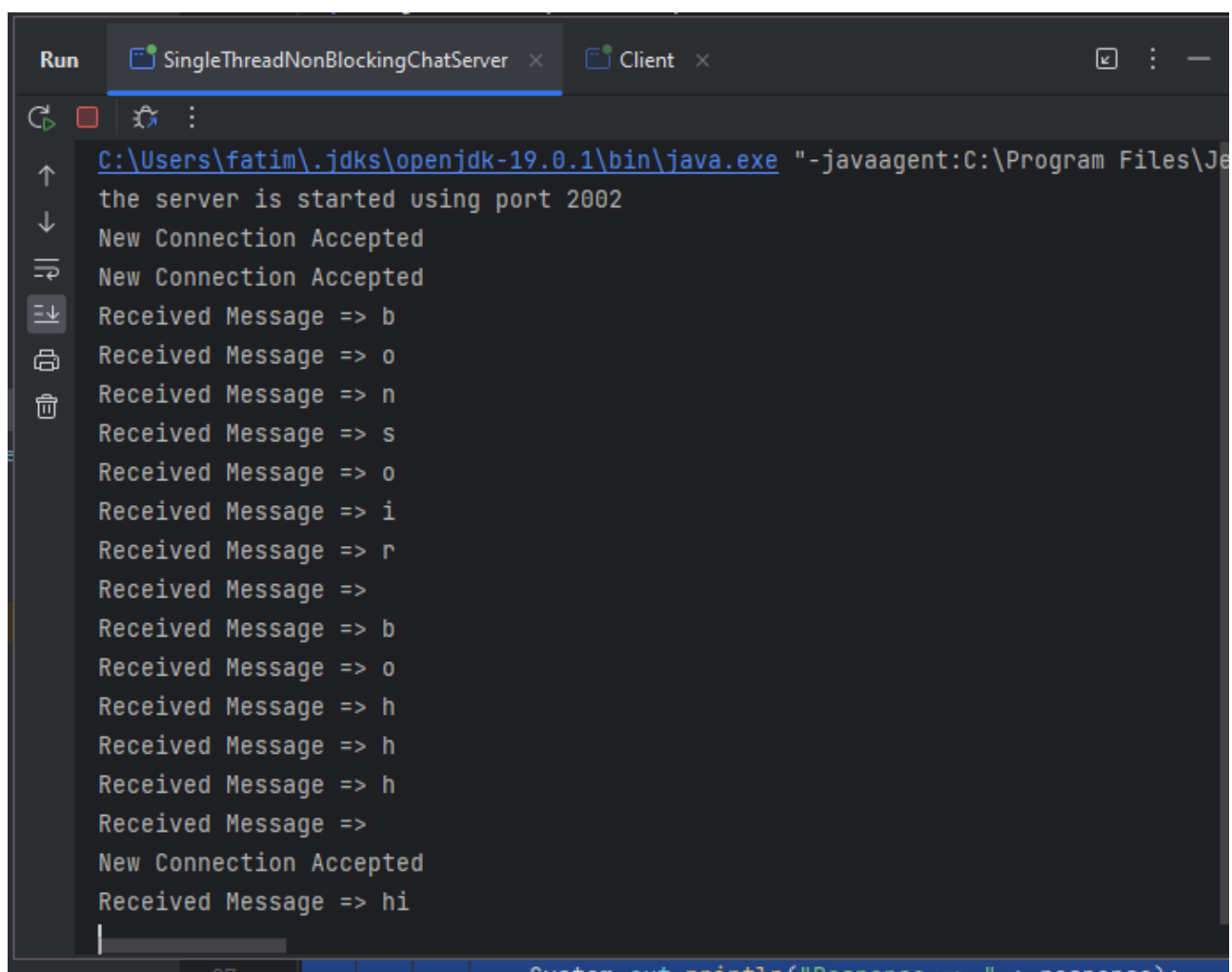
### Client python



```

"C:\Fatiza\S4\Architectures distribuées et Middlewares\Client Serveur Python\venv\Sc
hi

```



The screenshot shows a Java IDE with two tabs: "SingleThreadNonBlockingChatServer" and "Client". The "Run" console is active, displaying the output of the chat server. The output shows the server starting on port 2002, accepting multiple connections, and receiving messages. The messages received are: "b", "o", "n", "s", "o", "i", "r", an empty string, "b", "o", "h", "h", "h", an empty string, and "hi". The console also shows "New Connection Accepted" for the first three connections and "Received Message =>" for each subsequent message. The bottom of the console shows the start of a response line: "System.out.println("Response => " + response);".

```
Run | SingleThreadNonBlockingChatServer x | Client x
C:\Users\fatim\.jdk\openjdk-19.0.1\bin\java.exe "-javaagent:C:\Program Files\Je
the server is started using port 2002
New Connection Accepted
New Connection Accepted
Received Message => b
Received Message => o
Received Message => n
Received Message => s
Received Message => o
Received Message => i
Received Message => r
Received Message =>
Received Message => b
Received Message => o
Received Message => h
Received Message => h
Received Message => h
Received Message =>
New Connection Accepted
Received Message => hi
System.out.println("Response => " + response);
```

Au cours de ce TP sur les bases des systèmes distribués - programmation réseau en Java, j'ai appris comment établir une connexion à un serveur distant et envoyer des données à celui-ci à l'aide de la classe `SocketChannel`. J'ai également appris à utiliser les classes `Scanner` et `ByteBuffer` pour interagir avec les entrées/sorties et les tampons de données.

Grâce à cet apprentissage, j'ai acquis une compréhension fondamentale de la programmation réseau en Java, ce qui me permettra de construire des systèmes plus avancés et distribués en utilisant les bibliothèques et les architectures les plus récentes.