НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

КАФЕДРА АВТОМАТИКИ ТА УПРАВЛІННЯ В ТЕХНІЧНИХ СИСТЕМАХ

**Лабораторна робота № 1**

**з предмету «Програмування комп’ютерних та віртуальних мереж» на тему «ЗНАЙОМСТВО З TCP SOCKETS»**

Виконала:

студентка групи ІТ-з03мп

Козак О. С.

Перевірив:

Галушко Д.О.

**Київ – 2020р.**

**Лабораторна робота №1**

**Тема: знайомство з TCP sockets**

**Мета: ознайомитися з принципами роботи TCP sockets.**

**Завдання:** Реалізувати наступний клієнт-серверний застосунок:

**Сервер**:

Сервер має вміти «відповідати» на такі запити клієнта:

- Номер бригади;

- Прізвища усіх студентів у бригаді.

**Клієнт**:

Консольний застосунок, який демонструє функціональність сервера.

**Хід роботи:**

Для забезпечення обміну даними між клієнтом та сервером, ми використовуємо програмні інтерфейси – сокети. Сокети є кінцевими точками з'єднань в IP-мережах. Для роботи з TCP сокетами на мові програмування Java, ми використовуємо класи Socket та ServerSocket. Клас Socket оголошується на стороні клієнта, а сервер відтворює його, отримуючи сигнал на підключення. Так відбувається спілкування в мережі.

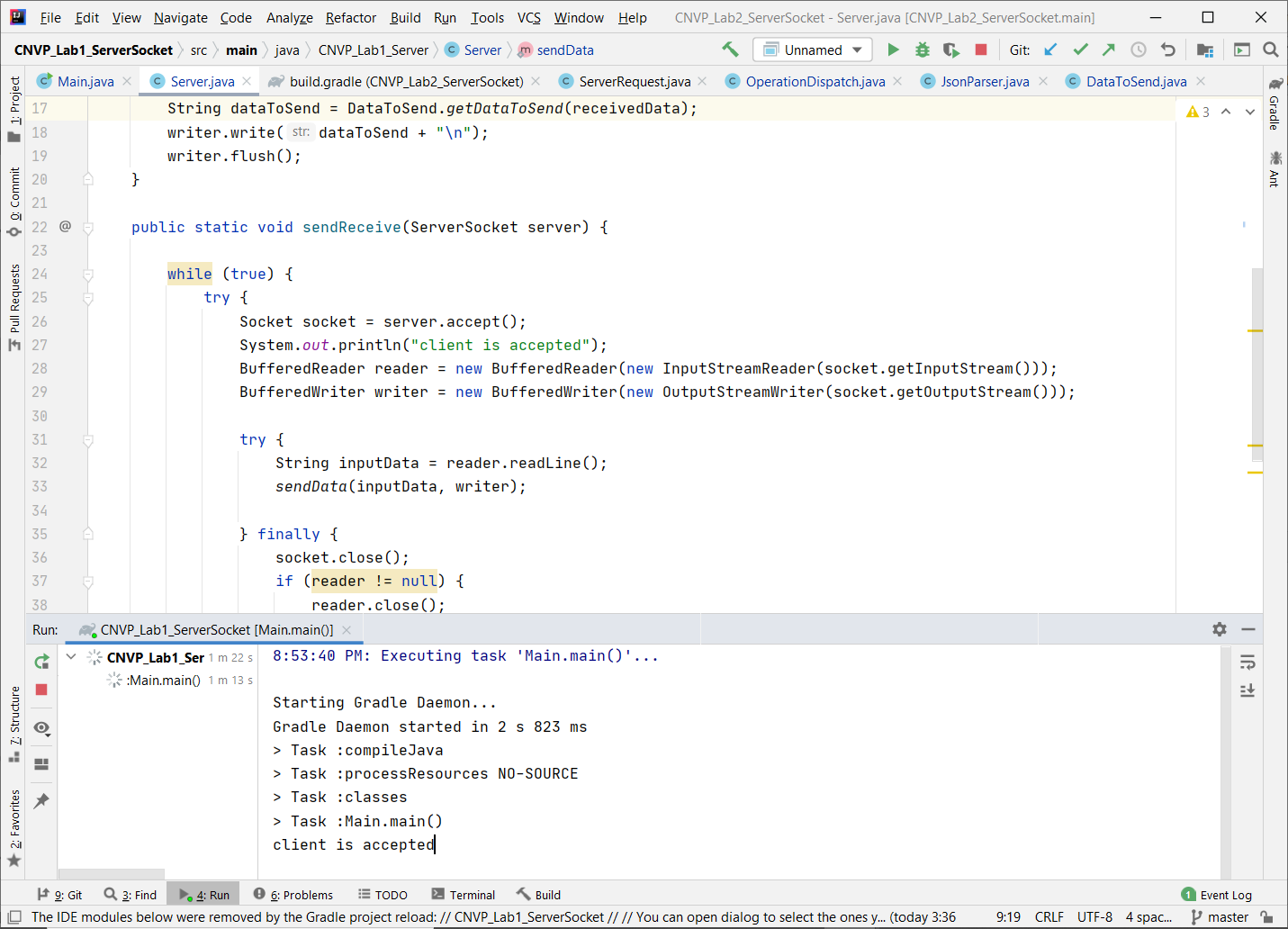
**1.** Створюємо сервер. Використовуємо клас ServerSocket, який реалізує серверний сокет, який очікує запити, які надходять від клієнтів по мережі, і може відправляти відповідь.

а) Створюємо клас Server.

**Лістинг Server:**

public class Server {  
 public static ServerSocket createServerSocket() throws IOException {  
 ServerSocket server;  
 InetAddress address = InetAddress.*getByName*("169.254.31.188");  
 server = new ServerSocket(5544, 50, address);  
 return server;  
 }  
  
 public static void sendData(String receivedData, BufferedWriter writer) throws IOException {  
 String dataToSend = DataToSend.*getDataToSend*(receivedData);  
 writer.write(dataToSend + "\n");  
 writer.flush();  
 }  
  
 public static void sendReceive(ServerSocket server) {  
  
 while (true) {  
 try {  
 Socket socket = server.accept();  
 System.*out*.println("client is accepted");  
 BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
 BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));  
  
 try {  
 String inputData = reader.readLine();  
 *sendData*(inputData, writer);  
  
 } finally {  
 socket.close();  
 if (reader != null) {  
 reader.close();  
 }  
 if (writer != null) {  
 writer.close();  
 }  
 }  
 } catch (IOException exception) {  
 exception.printStackTrace();  
 }  
 }  
 }  
}

Клас Server відповідає за створення сокету, надсилання та отримання даних. У методі *createServerSocket* ми створюємо сокет сервера, надаємо йому значення порта та IP-адреси. У методі *sendReceive* ми створюємо екземпляр класу Socket для відтворення з’єднання з клієнтом та з допомогою методу accept, отримує запит на підключення від клієнта:



б) Створюємо екземпляри класу BufferedReader та BufferedWriter для зчитування даних з буферу, де вони зберігаються після зчитування з потоку. Передаючи InputStreamReader та OutputStreamWriter в конструктор, ми вказуємо на зчитування даних з потоку та записування в потік.

З допомогою метода *readLine*() зчитуємо запит від клієнта.

в) Розглянемо метод *sendData*, який відповідає за відсилання відповіді клієнту. У методі *sendData* ми викликаємо метод *getDataToSend* з класу DataToSend.

**Лістинг DataToSend:**

package CNVP\_Lab1\_Server;

import java.io.IOException;  
   
public class DataToSend {  
 public static String getDataToSend(String receivedData) throws IOException {  
 String data = OperationDispatch.*getOperation*(receivedData);  
 if (data.equals("GetNumberOfMyBrigade")) {  
 return OperationDispatch.*getBrigadeNumber*();  
 } else if (data.equals("GetSurnamesOfMyBrigade")) {  
 return OperationDispatch.*getSurnames*();  
 } else {  
 throw new UnsupportedOperationException();  
 }  
 }  
}

г) Від клієнта серверу приходять два запити: отримання номеру бригади, отримання прізвища всіх студентів у бригаді. Для отримання відповіді на один із запитів ми використовуємо клас OperationDispatch, який визначає, який сам запит прийшов та які дані повинні бути відправленні у відповідь. Метод *getOperation* вказує, який саме запит прийшов від клієнта. Методи *getBrigadeNumber* та *getSurnames* надають номер бригади та прізвищавсіх студентів бригади відповідно*.*

**Лістинг OperationDispatch:**

package CNVP\_Lab1\_Server;  
  
import com.fasterxml.jackson.core.JsonProcessingException;  
import java.io.IOException;  
   
public class OperationDispatch {  
 public static String getOperation(String receivedData) throws IOException {  
 return JsonParser.*convertFromJson*(receivedData);  
 }  
  
 public static String getBrigadeNumber() throws JsonProcessingException {  
 String numberOfBrigade = *getNumber*();  
 return JsonParser.*convertToJson*(numberOfBrigade);  
 }  
  
 public static String getNumber() {  
 return "2";  
 }  
  
 public static String getSurnames() throws JsonProcessingException {  
 String surnames = String.*join*(" ", "ПОЛІЩУК", "ШЕВЧУК", "ПОПОВА", "ХАЧАТУРЯН", "СЕРДЮК");  
 return JsonParser.*convertToJson*(surnames);  
 }  
}

д) Дані у нашому додатку, які надсилаються по сокетам, представлені у вигляді Json. Для серіалізації та десеріалізації, використовуючи екземпляр класу ObjectMapper.

**Лістинг JsonParser:**

package CNVP\_Lab1\_Server;  
  
import com.fasterxml.jackson.core.JsonProcessingException;  
import com.fasterxml.jackson.databind.ObjectMapper;  
  
import java.io.IOException;  
   
  
public class JsonParser {  
  
 public static String convertToJson(String message) throws JsonProcessingException {  
 ObjectMapper objectMapper = new ObjectMapper();  
 ServerRequest serverRequest = new ServerRequest();  
 serverRequest.setOperation(message);  
 return objectMapper.writeValueAsString(serverRequest);  
 }  
  
 public static String convertFromJson(String received) throws IOException {  
 ObjectMapper objectMapper = new ObjectMapper();  
 ServerRequest serverRequest = objectMapper.readValue(received, ServerRequest.class);  
 String result = serverRequest.getOperation();  
 return result;  
 }  
}

е) Таким чином ми отримуємо відповідь на запит клієнта та з допомогою методу *write* та *flush* у методі sendData класу Server надсилаємо її клієнту. Після цього закриваємо сокет використовуючи метод *close*.

є) У класі Main ми створюємо сокет сервера та викликаємо метод *sendReceive*.

**Лістинг Main:**

package CNVP\_Lab1\_Server;  
  
import java.io.IOException;  
import java.net.ServerSocket;  
   
  
  
public class Main {  
 public static void main(String[] args) {  
 try {  
 ServerSocket server = Server.*createServerSocket*();  
 Server.*sendReceive*(server);  
 } catch (IOException exception) {  
 exception.printStackTrace();  
 }  
 }  
}

2. Створюємо клієнт.

а) В класі Client створюємо метод *createSocket* для створення сокету клієнта.

**Лістинг Client:**

package CNVP\_Lab1\_Client;  
  
import java.io.\*;  
import java.net.InetAddress;  
import java.net.Socket;  
  
public class Client {  
 static int *port* = 5544;  
  
 public static Socket createSocket(InetAddress ipAddressOfServer) throws IOException {  
 Socket clientSocket = new Socket(ipAddressOfServer, *port*);  
 if (clientSocket.isConnected()) {  
 System.*out*.println("Connected to server"+"\n");  
 }  
 return clientSocket;  
 }  
  
 public static void sendReceive(Socket clientSocket, String input) {  
 BufferedReader reader = null;  
 BufferedWriter writer = null;  
 try {  
 try {  
 writer = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()));  
 reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));  
 SendReceive.*sendDataToServer*(input, writer);  
 String receiveMessage = SendReceive.*receive*(reader);  
 System.*out*.println(OperationDispatch.*getConsoleOut*(input, receiveMessage));  
 } finally {  
 if (reader != null) reader.close();  
 else {  
 System.*out*.println("reader is null");  
 }  
 if (writer != null) writer.close();  
 else {  
 System.*out*.println("writer is null");  
 }  
 }  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

Вказуючи IP-адресу серверу та порт, підключаємось до сервера. Метод *sendReceive* відправляє запити на сервер та приймає відповіді сервера. Створюємо екземпляри класу BufferedReader та BufferedWriter.

б) Клієнт робить два запити – на номер бригади та на прізвища студентів в бригаді. Для показу опцій та зчитування запиту користувача використовуємо метод *readUserInput* класу UserInput.

**Лістинг UserInput:**

import java.net.InetAddress;  
import java.net.UnknownHostException;  
import java.util.Scanner;  
  
public class UserInput {  
 public static void displayOptions() {  
 System.*out*.println("get number of brigade");  
 System.*out*.println("get surnames of brigade");  
 System.*out*.println("exit");  
 }  
  
 public static boolean tryParseInt(String value) {  
 try {  
 Integer.*parseInt*(value);  
 return true;  
 } catch (NumberFormatException e) {  
 return false;  
 }  
 }  
  
 public static String readUserInput() {  
 *displayOptions*();  
 Scanner in = new Scanner(System.*in*);  
 while (true) {  
 String input = in.nextLine();  
 if (input.equals("get number of brigade") || input.equals("get surnames of brigade") || input.equals("exit")) {  
 return input;  
 } else {  
 System.*out*.println("Input is not correct");  
 *displayOptions*();  
 }  
 }  
 }  
  
 public static InetAddress getInetAddress() throws UnknownHostException {  
 System.*out*.println("Input server's IP address or use default 127.0.0.1");  
 Scanner in = new Scanner(System.*in*);  
 while (true) {  
 String ipAddressOfServer = in.nextLine();  
 boolean validation = *validateIp*(ipAddressOfServer);  
 if (validation) {  
 return InetAddress.*getByName*(ipAddressOfServer);  
 } else {  
 System.*out*.println("Input server's IP address or use default 127.0.0.1");  
 }  
 }  
 }  
  
 public static boolean validateIp(String inputedIp) {  
 boolean validation = false;  
 if (Validation.*isValidInetAddress*(inputedIp)) {  
 return true;  
 } else {  
 System.*out*.print("The IP address " + inputedIp + " isn't valid");  
 return false;  
 }  
 }  
}

в) Клас SendReceive відповідає за відправляння відповідного до введеного користувачем запиту на сервер.

package CNVP\_Lab1\_Client;  
  
import com.fasterxml.jackson.core.JsonProcessingException;  
  
import java.io.BufferedReader;  
import java.io.BufferedWriter;  
import java.io.IOException;  
  
public class SendReceive {  
 public static String getDataToSend(String input) throws JsonProcessingException {  
 String operation = OperationDispatch.*getOperation*(input);  
 String converted = JsonParser.*convertToJson*(operation);  
 return converted;  
 }  
  
 public static void sendDataToServer(String input, BufferedWriter writer) throws IOException {  
 String dataToSend = *getDataToSend*(input);  
 writer.write(dataToSend + "\n"); *//send* writer.flush();  
 }  
  
 public static String receive(BufferedReader reader) throws IOException {  
 String serverMassage = reader.readLine();  
 return JsonParser.*convertFromJson*(serverMassage);  
 }  
}

г) Клас OperationDispatchзаймається створенням відповідного запиту.

**Лістинг OperationDispatch**

package CNVP\_Lab1\_Client;  
  
import java.io.IOException;  
  
public class OperationDispatch {  
 public static String getOperation(String input) {  
  
 if (input.equals("get number of brigade")) {  
 return "GetNumberOfMyBrigade";  
 } else if (input.equals("get surnames of brigade")) {  
 return "GetSurnamesOfMyBrigade";  
 } else {  
 throw new UnsupportedOperationException();  
 }  
 }  
  
 public static String getConsoleOut(String input, String receivedData) throws IOException {  
 if (input.equals("get number of brigade")) {  
 return "number of my brigade: " + receivedData;  
 } else if (input.equals("get surnames of brigade")) {  
 return "surnames of my brigade: " + receivedData;  
 } else {  
 throw new IOException("Invalid number of operation");  
 }  
 }  
}

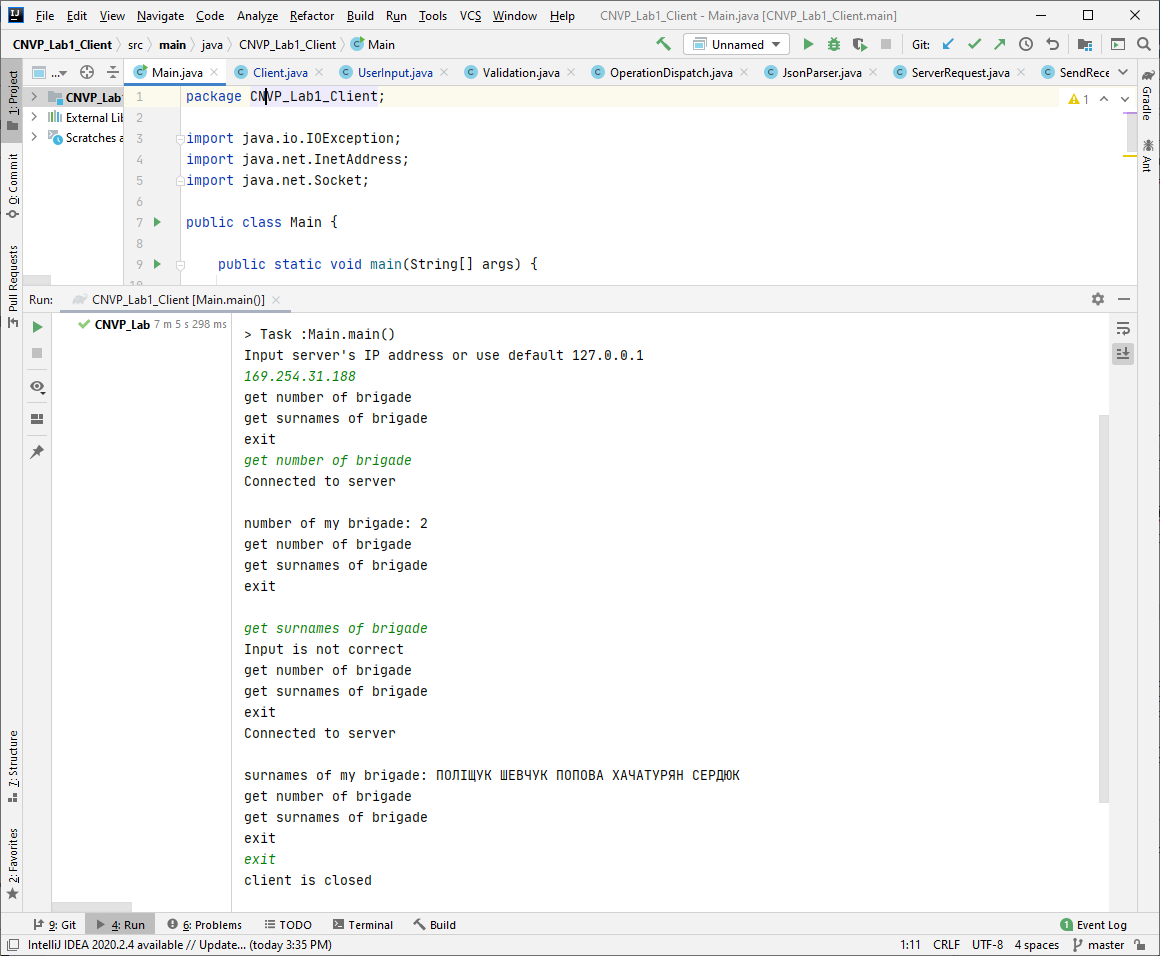
в) Так само як і на сервері, запити та відповіді представленні у вигляді Json, серіалізацією та десеріалізацією займається клас JsonParser.

У класі Main створюємо сокет, вводимо Ip-адресу сервера, підключаємось до сервера, зчитуємо запит за викликаємо метод *sendReceive* класу Client.

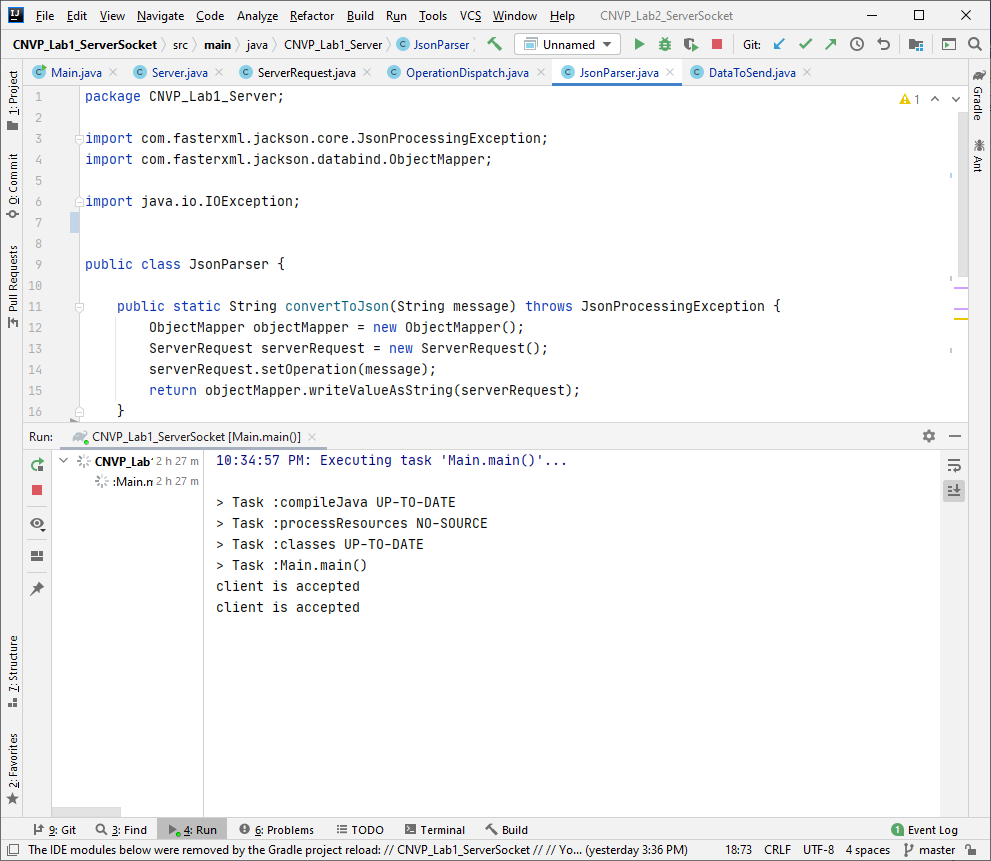
**Лістинг Main:**

package CNVP\_Lab1\_Client;  
  
import java.io.IOException;  
import java.net.InetAddress;  
import java.net.Socket;  
  
public class Main {  
  
 public static void main(String[] args) {  
  
 String input = null;  
 Socket socketClient = null;  
 try {  
 try {  
 InetAddress ipAddressOfServer = UserInput.*getInetAddress*();  
 while (true) {  
 input = UserInput.*readUserInput*();  
 if (input.equals("exit")) {  
 break;  
 }  
 socketClient = Client.*createSocket*(ipAddressOfServer);  
 Client.*sendReceive*(socketClient, input);  
 }  
 } finally {  
 if (socketClient != null) {  
 socketClient.close();  
 System.*out*.println("client is closed");  
 }  
 }  
 } catch (IOException ex) {  
 ex.printStackTrace();  
 }  
 }  
}

4. Результат роботи клієнта та виконання запитів:



5. Результат роботи сервера:



Висновки:

Я ознайомилась з принципами роботи TCP-сокетів та розробила клієнт-серверний додаток, який відповідає на такі запити: отримати номер бригади; отримати прізвище всіх студентів у бригаді.