МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ

Кафедра системного проектування

**ЛАБОРАТОРНА РОБОТА №5**

з дисципліни: «ПАРАЛЕЛЬНІ ОБЧИСЛЕННЯ»

Виконав:

студент 3 курсу

групи ДА-81

Головін Б. О.

Варіант 5

Київ – 2021

***Лабораторна робота 5***

**Дослідження мережевих можливостей розподіленних систем та реалізація клієнт-серверного додатку.**

***Мета роботи:***

Дослідження мережевих можливостей розподіленних систем та реалізація клієнт-серверного додатку.

***Завдання*** Варіант 5

Розробити клієнт-серверний додаток, використовуючи Java RMI, Sockets або аналогічну технологію.

**5)** Сервер розсилає повідомлення обраним зі списку клієнтам. Список зберігається в файлі.

**Лістинг програми**

**Server.java**

import java.io.\*;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.util.LinkedList;  
  
public class Server {  
  
 public static final int *PORT* = 8080;  
 public static LinkedList<ServerHelper> *serverList* = new LinkedList<>(); // список всех сокетов связаных с клиентом  
  
 public static void main(String[] args) throws IOException {  
 try (ServerSocket serverSocket = new ServerSocket(*PORT*)) { // создание сервера  
 System.*out*.println("Server is running");  
 new ReadConsole().start(); // поток для чтения с консоли в сервере  
 while (true) {  
 Socket clientSocket = serverSocket.accept();  
 try {  
 *serverList*.add(new ServerHelper(clientSocket)); // добавить новое соединенние в список  
 } catch (IOException e) {  
 clientSocket.close();  
 }  
 }  
 }  
 }  
}

**ServerHelper.java**

import java.io.\*;  
import java.net.Socket;  
  
class ServerHelper extends Thread {  
  
 private Socket socket; // сокет, через который сервер общается с клиентом  
 private BufferedReader reader; // поток для чтения с консоли  
 private BufferedReader in; // поток чтения из сокета  
 private BufferedWriter out; // поток записи в сокет  
 private String date; // время и дата присоеденинения  
 public String nickname; // никнейм подключенного клиента  
  
 public ServerHelper(Socket socket) throws IOException {  
 this.socket = socket;  
 reader = new BufferedReader(new InputStreamReader(System.*in*));  
 in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
 out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));  
 start(); // вызываем run()  
 }  
  
 @Override  
 public void run() {  
 try {  
 date = in.readLine();  
 nickname = in.readLine();  
 System.*out*.println("new Client - " + nickname + " - " + date);  
 send("new client is registered");  
 send("\n --- Wait message from server --- \n");  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 void send(String msg) { // метод для удобной отправки сообщений  
 try {  
 out.write(msg + "\n");  
 out.flush();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

**ReadConsole.java**

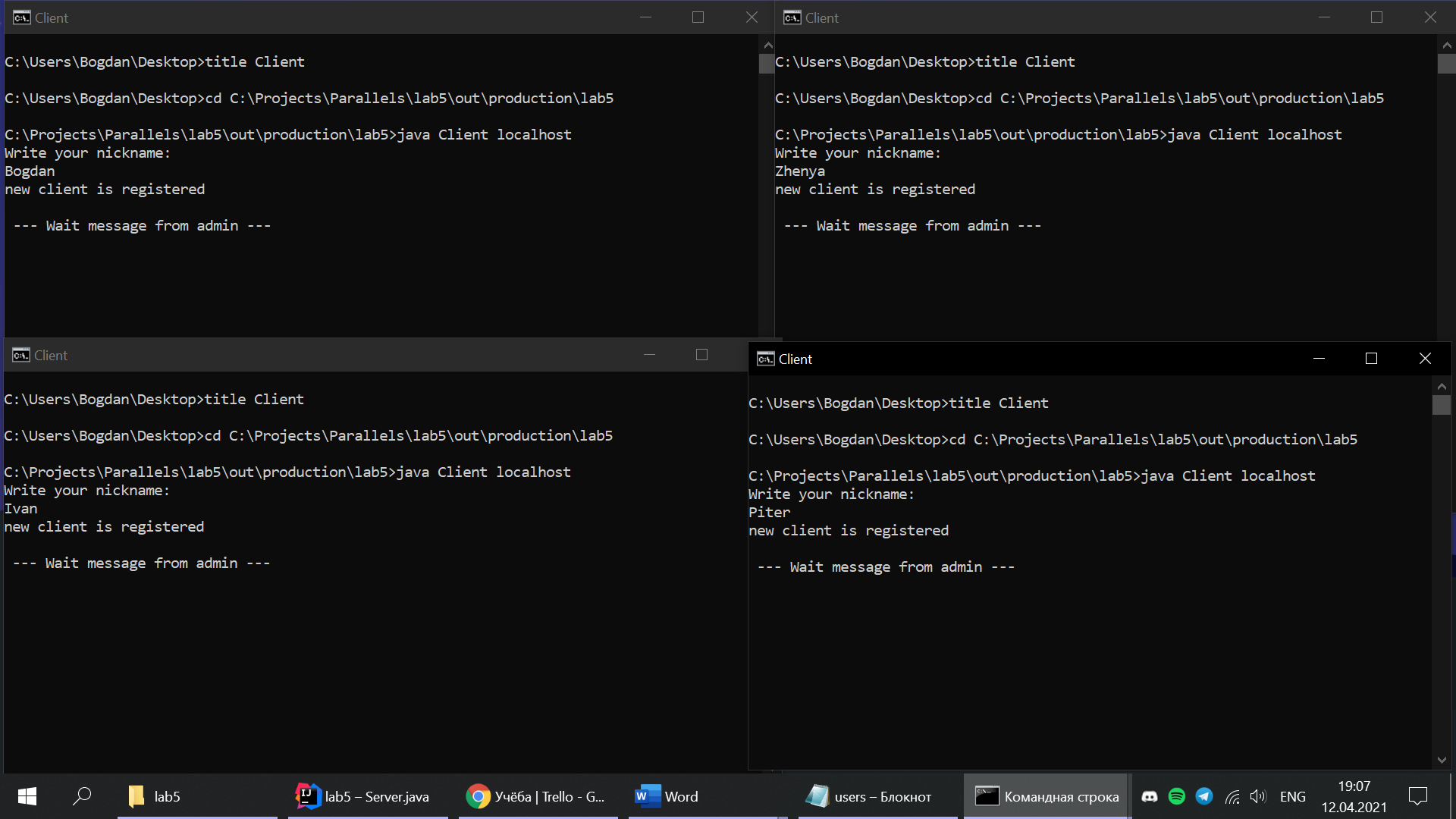
import java.io.\*;  
  
class ReadConsole extends Thread {  
  
 private BufferedReader reader = new BufferedReader(new InputStreamReader(System.*in*));  
  
 @Override  
 public void run() {  
  
 String command;  
 try {  
 while (true) {  
 command = reader.readLine();  
 if (command.equals("take")) { // записуем список пользователей в файл для удобной обработки  
 try (FileWriter writer = new FileWriter("C:\\Users\\Bogdan\\Desktop\\users.txt", false)) {  
 for (ServerHelper user : Server.*serverList*) {  
 String text = user.nickname;  
 writer.write(text + "\n");  
 }  
 writer.flush();  
 System.*out*.println("File users.txt is already filled with data");  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 } else if (command.equals("push")) {// после ручного отредактирования файла // отсылаем нужным пользователям сообщение  
 try {  
 File file = new File("C:\\Users\\Bogdan\\Desktop\\users.txt");  
 FileReader fr = new FileReader(file);  
 BufferedReader readerFile = new BufferedReader(fr);  
 String line = readerFile.readLine();  
 while (line != null) {  
 System.*out*.println(line);  
 for (ServerHelper user : Server.*serverList*) {  
 if (user.nickname.equals(line))  
 user.send("hello " + line);  
 }  
 line = readerFile.readLine();  
 }  
 System.*out*.println("push completed");  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

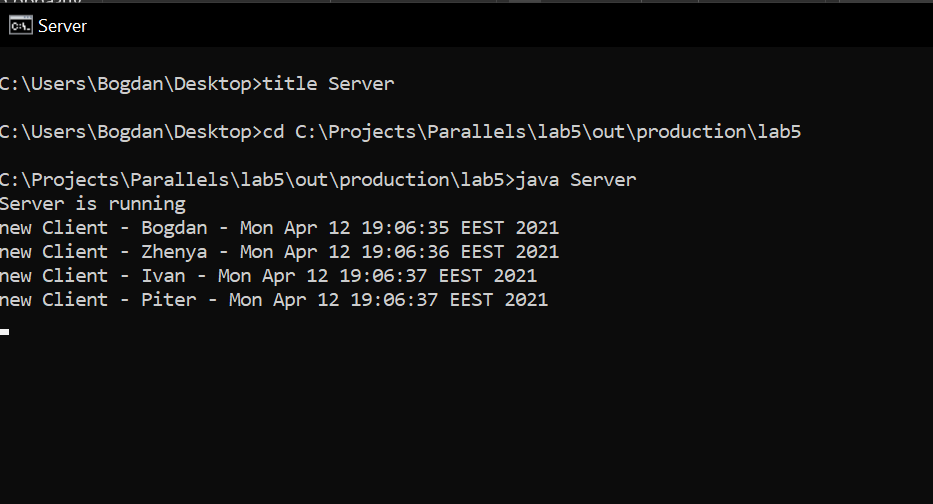
**Client.java**

import java.io.\*;  
import java.net.Socket;  
import java.util.Date;  
  
public class Client {  
  
 private static Socket *clientSocket*; // сокет для общения  
 private static BufferedReader *reader*; // поток для чтения с консоли  
 private static BufferedReader *in*; // поток чтения из сокета  
 private static BufferedWriter *out*; // поток записи в сокет  
 private static String *date*; // время подключения клиента  
 private static String *nickname*; // никнейм подключенного клиента  
  
 public static void main(String[] args) {  
 try {  
 try {  
 *clientSocket* = new Socket("localhost", 8080); // запрашиваем у сервера доступ на соединение  
 *reader* = new BufferedReader(new InputStreamReader(System.*in*));  
 *in* = new BufferedReader(new InputStreamReader(*clientSocket*.getInputStream()));  
 *out* = new BufferedWriter(new OutputStreamWriter(*clientSocket*.getOutputStream()));  
  
 *date* = new Date().toString();  
 System.*out*.println("Write your nickname:");  
 *nickname* = *reader*.readLine();  
  
 *send*(*date*);  
 *send*(*nickname*);  
  
 String serverWord;  
 while (true){ // постоянно смотрим на входящие данные с сервера и если они есть, выводим  
 serverWord = *in*.readLine();  
 if (serverWord != null){  
 System.*out*.println(serverWord);  
 }  
 }  
 } finally { // в любом случае необходимо закрыть сокет и потоки  
 System.*out*.println("Клиент был закрыт...");  
 *clientSocket*.close();  
 *in*.close();  
 *out*.close();  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private static void send(String msg) {  
 try {  
 *out*.write(msg + "\n");  
 *out*.flush();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

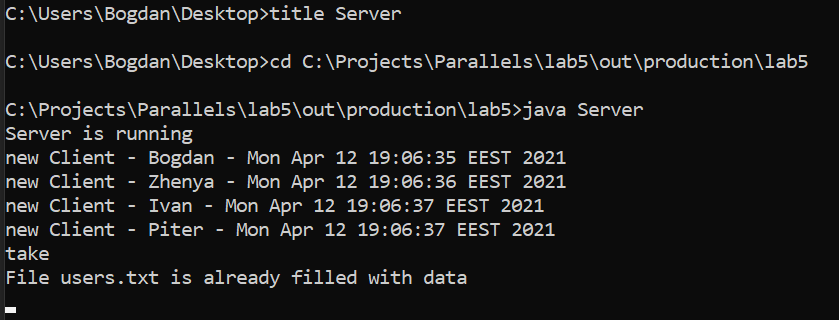
**Результати роботи**

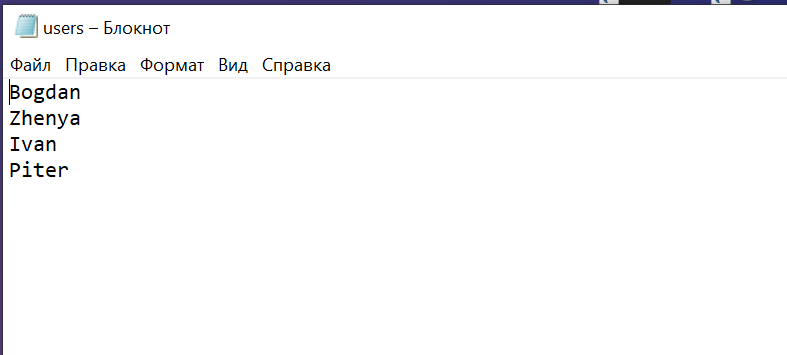
Для перевірки запустимо 1 сервер та 4 кліента. Кожний клієнт зайде під ім’ям. Таким чином на сервері з’являться повідомлення про їх під’єднання



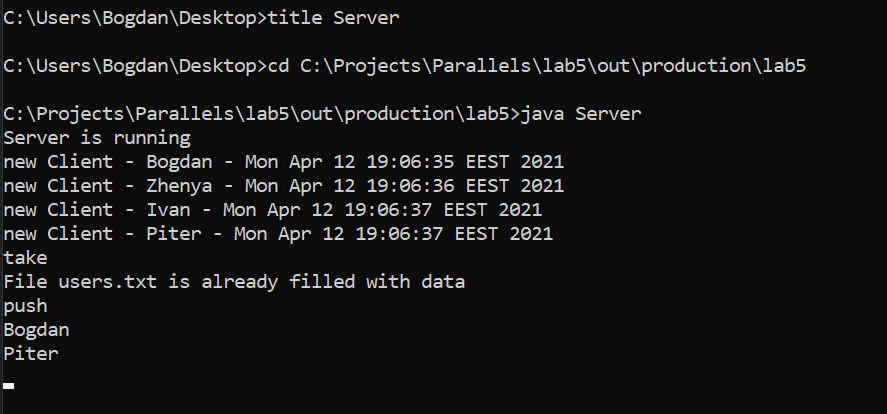
****

Тепер треба чи самому прописати імена яким ми хочемо відправити, чи можемо скористатися спеціально прописаною командою *take*у сервері, яка згенерує текстовий файл з логінами клієнтів і видалити зайві.

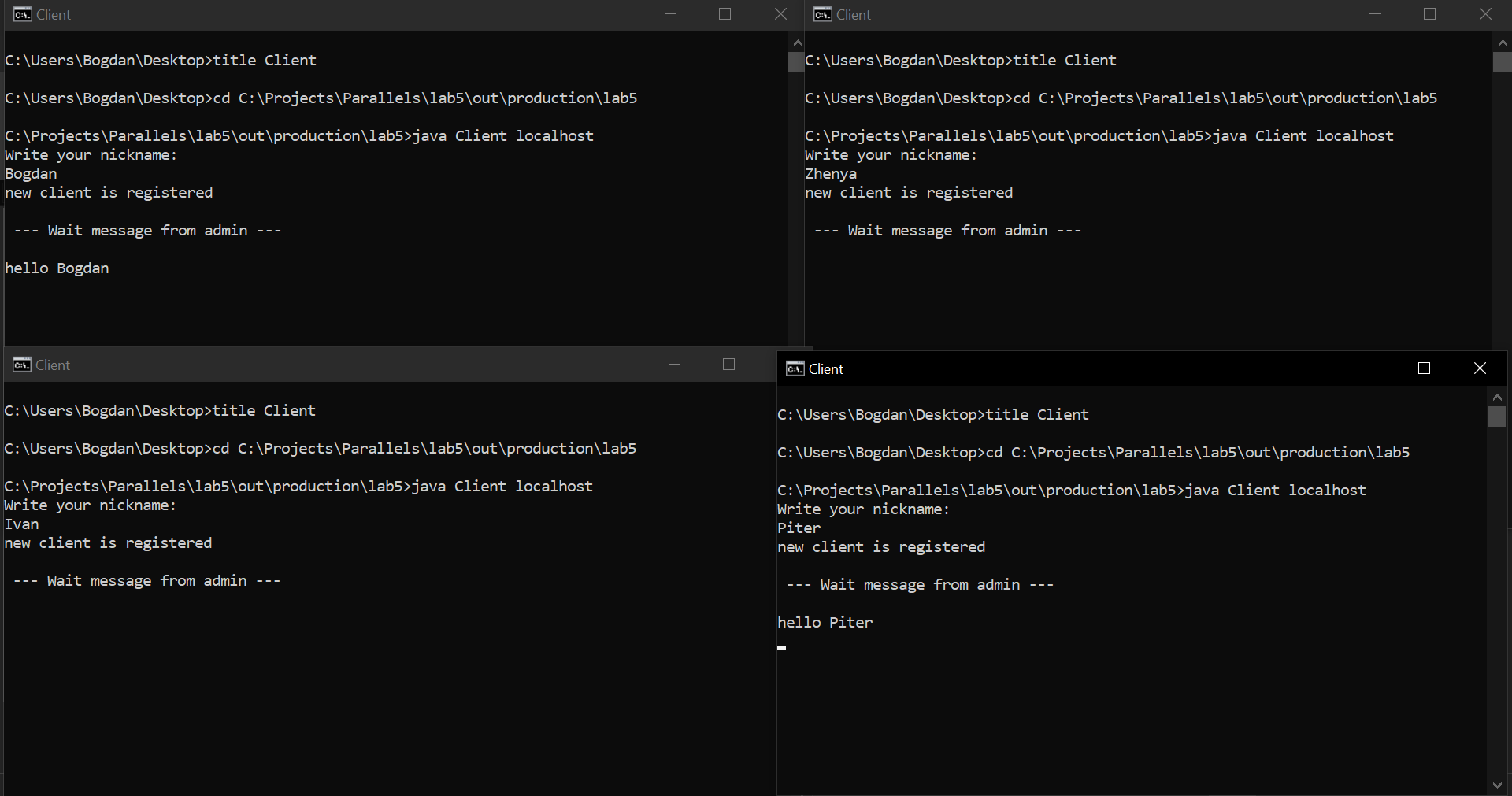
****

****

Видалимо кілька імен з цього файлу та розішлемо повідомлення через команду *push* у сервері. У сервері покажуться імена яким відправлено повідомлення.

****

Зайдемо у клієнти і подивимось.

****

Як бачимо програма працює коректно, повідомлення відправилось потрібним клієнтам.

**Опис роботи**

**Server.java**

У даному класі ми створюємо сервер, а також під’єднуємо до нього клієнтів. Сервер та клієнт можуть спілкуватися тільки через сокет. Щоб мати можливість зроби программу на багато користувачів треба сокети через які під’єднані клієнти записати у список.

Також тут запускається потік для зчитування команд з консолі *new ReadConsole().start();*

**ReadConsole.java**

За допомогою цього класу реалізовується управління повідомленнями які відправляє сервер.

Командою *take* він бере логіни користувачів та записує у файл (щоб було простіше потім їх редагувати).

Командою *push* він відправляє повідмлення обраним користувачам (користувачі записані у файлі).

**ServerHelper.java**

У цьому класі реалізовується зв’язок одного клієнта з сервером. При підключенні клієнт дає дані про час підключення та його логін. У свою чергу сервер говорить про те що підключення відбулося.

Також реалізован метод *send()* для зручності відправлень повідомлень.

**Client.java**

Спочатку клієнт підключається за вказаним портом та хостом. Далі заповнює логін та відсилає на сервер дані (логін, час підключення).

Далі він постійно очікує повідомлення з сервера. Якщо воно прийде воно відобразиться у программі.

**Висновки**

У даній лабораторній роботі ми ознайомитись з дослідженням мережевих можливостей розподіленних систем та реалізацією клієнт-серверного додатку.

Було створено клієнт-серверний додаток за допомогою Java Sockets, який реалізовував задачу:

Сервер розсилає повідомлення обраним зі списку клієнтам. Список зберігається в файлі.