МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОСИЙСКОЙ ФЕДЕРАЦИИ

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

**Лабораторная работа №4**

**По дисциплине «Технология программирования»**

**на тему**

**Сетевые «клиент-серверные» приложения**

Группа: АВТ-808

Студент: Попов Д.А.

Вариант: 7

Новосибирск, 2020

***Задание:***

1. Изучить особенности реализации сетевых приложений в Java.

2. Доработать программу, созданную в лабораторной работе № 5:

1) создать отдельное консольное приложение, которое будет играть роль TCP-сервера. Клиентом будет приложение, которое было создано в предыдущих работах;

2) TCP-сервер должен ожидать подключения клиентов и выдавать вновь подключенному клиенту список уже подключенных. Необходимо также внести изменения в интерфейс клиента, так чтобы в панели управления отображался список всех подключенных к серверу клиентов. При отсоединении клиентов или при подключении новых список должен обновляться;

3) запрограммировать специальное взаимодействие по TCP с другими клиентами через сервер по варианту;

Вариант 7

Реализовать возможность скопировать и добавить в симуляцию все объекты одного из подключенных клиентов.

***Описание доработки программы:***

На стороне клиента создан класс Client, который отвечает за все взаимодействие с сервером, а именно: высылает запрос на добавление подключенного пользователя к списку всех пользователей, высылает запрос на получение списка пользователей, высылает запрос на получение объектов другого пользователя, высылает запрос на исключение пользователя из общего списка при отключении.

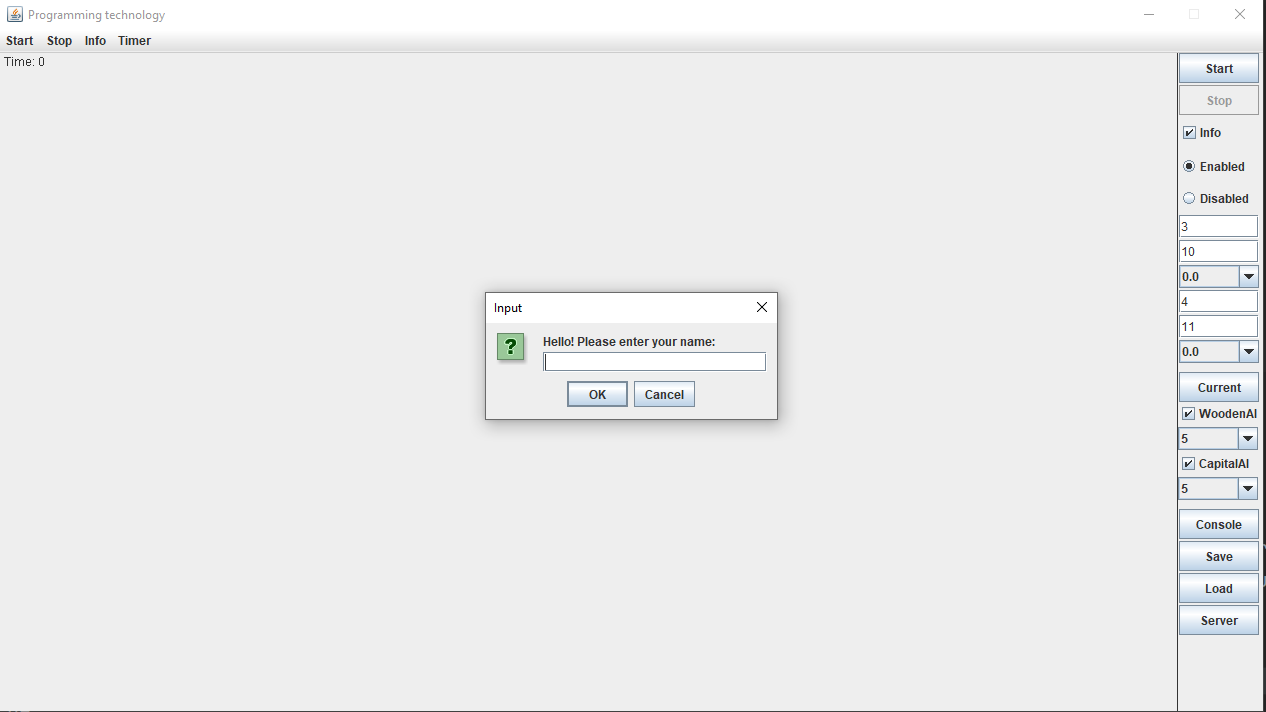
public class Client {  
 private static ArrayList<String> *users* = new ArrayList<>();  
 private static String *username*;  
 private static Socket *socket*;  
  
 private static final String *SET\_USER* = "setUser";  
 private static final String *GET\_USERS* = "getUsers";  
 private static final String *GET\_OBJECTS* = "getObjects";  
 private static final String *DISCONNECT* = "disconnect";  
  
 public Client() {  
 *username* = JOptionPane.*showInputDialog*("Hello! Please enter your name:");  
 try {  
 *socket* = new Socket("localhost", 6141);  
 ObjectOutputStream objectOutputStream = new ObjectOutputStream(*socket*.getOutputStream());  
 objectOutputStream.writeUTF(*username*);  
 objectOutputStream.writeUTF(*SET\_USER*);  
 objectOutputStream.writeUTF("targetUser");  
 objectOutputStream.writeObject(null);  
 objectOutputStream.flush();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public static ArrayList<String> getUsers() {  
 try{  
 *socket* = new Socket("localhost", 6141);  
 ObjectOutputStream objectOutputStream = new ObjectOutputStream(*socket*.getOutputStream());  
 objectOutputStream.writeUTF(*username*);  
 objectOutputStream.writeUTF(*GET\_USERS*);  
 objectOutputStream.writeUTF("targetUser");  
 objectOutputStream.writeObject(Singleton.*getSingleton*().getHouseVector());  
 objectOutputStream.flush();  
  
 ObjectInputStream objectInputStream = new ObjectInputStream(*socket*.getInputStream());  
 *users* = (ArrayList<String>) objectInputStream.readObject();  
 objectInputStream.close();  
  
 } catch (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 return *users*;  
 }  
  
 public static void getHouses(){  
 String targetUser = "";  
 String s;  
 try{  
 while (!*users*.contains(targetUser)){  
 targetUser = JOptionPane.*showInputDialog*("Please enter target name:");  
 }  
 *socket* = new Socket("localhost", 6141);  
 ObjectOutputStream objectOutputStream = new ObjectOutputStream(*socket*.getOutputStream());  
 objectOutputStream.writeUTF(*username*);  
 objectOutputStream.writeUTF(*GET\_OBJECTS*);  
 objectOutputStream.writeUTF(targetUser);  
 objectOutputStream.writeObject(Singleton.*getSingleton*().getHouseVector());  
 System.*out*.println(targetUser);  
 objectOutputStream.flush();  
  
 ObjectInputStream objectInputStream = new ObjectInputStream(*socket*.getInputStream());  
 Singleton.*getSingleton*().clearArea();  
 Singleton.*getSingleton*().setHouseVector((Vector<House>) objectInputStream.readObject());  
 MyComponent.*setCount*(Singleton.*getSingleton*().getHouseVector().size());  
 for(int i = 0; i < Singleton.*getSingleton*().getHouseVector().size(); i++){  
 Singleton.*getSingleton*().getHouse(i).setTimeOfBirth(MyComponent.*getTime*());  
 }  
 objectInputStream.close();  
  
 } catch (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public static void disconnect(){  
 try {  
 *socket* = new Socket("localhost", 6141);  
 ObjectOutputStream objectOutputStream = new ObjectOutputStream(*socket*.getOutputStream());  
 objectOutputStream.writeUTF(*username*);  
 objectOutputStream.writeUTF(*DISCONNECT*);  
 objectOutputStream.writeUTF("targetUser");  
 objectOutputStream.writeObject(null);  
 objectOutputStream.flush();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

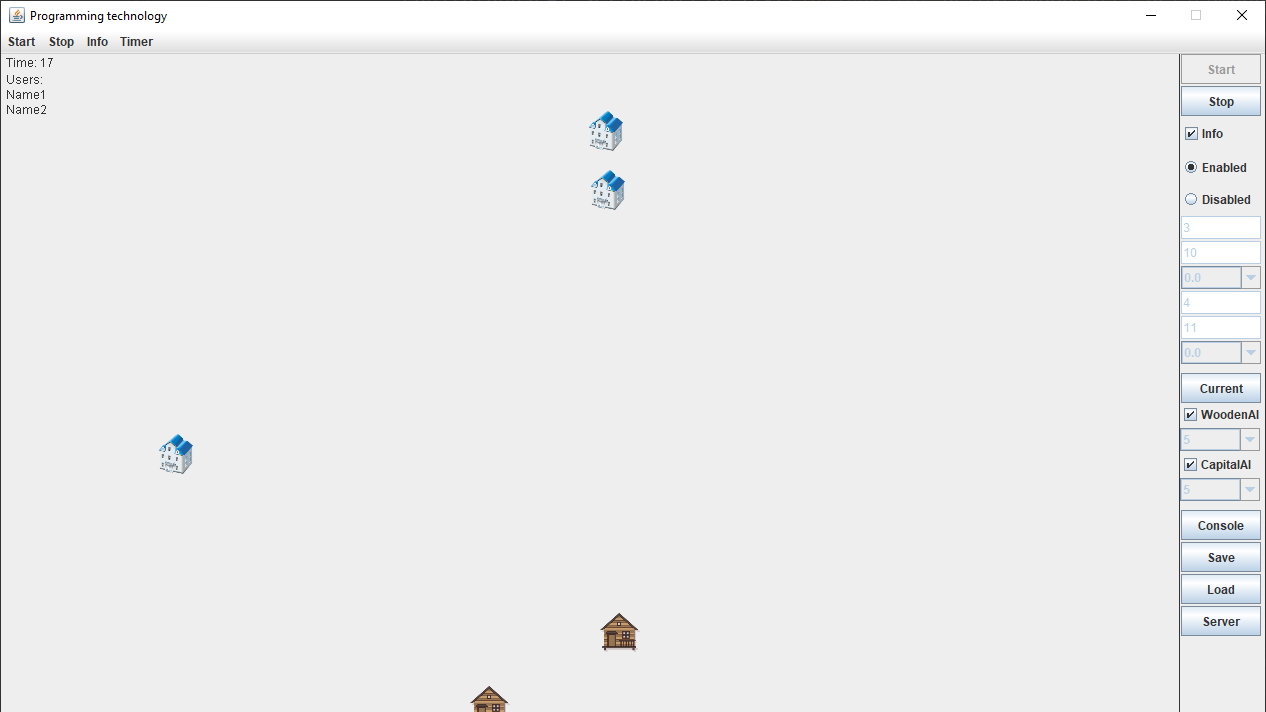
На стороне сервера создан класс ServerMain, который ожидает подключения нового пользователя и выделяет для него поток.

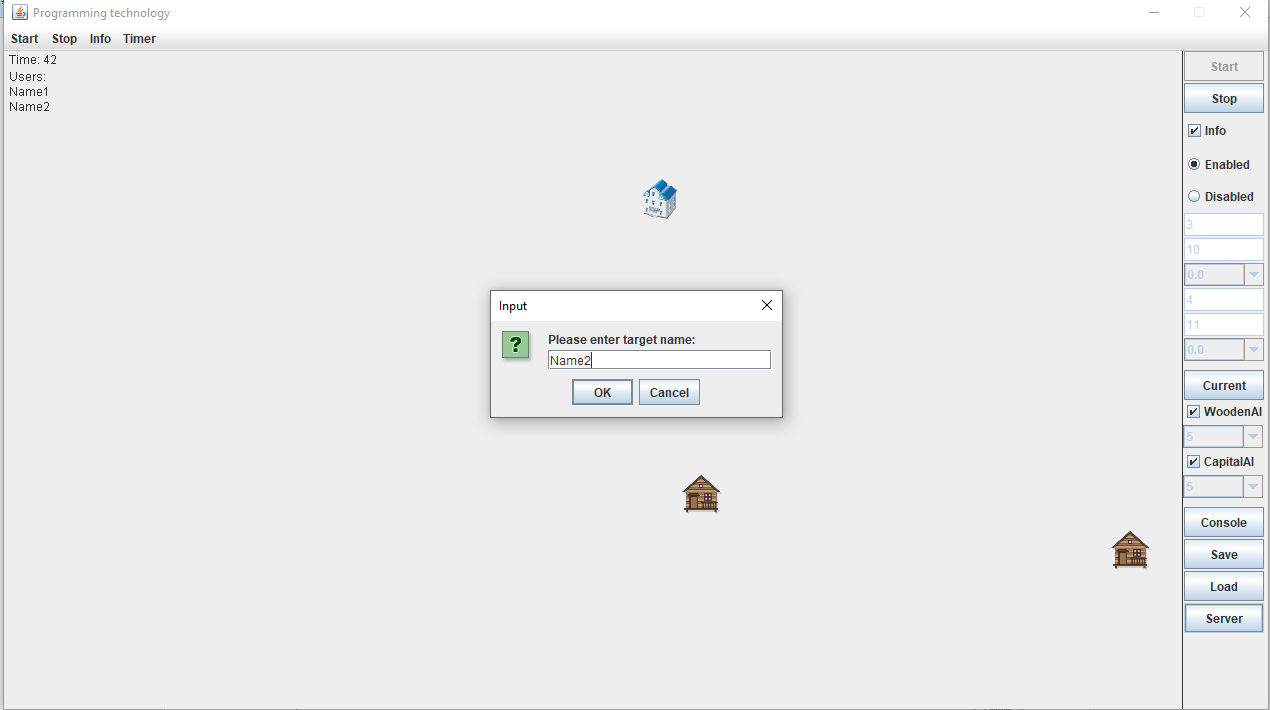
public class ServerMain {  
 public static void main(String[] args) throws IOException {  
 try(ServerSocket serverSocket = new ServerSocket(6141)){  
 while (true){  
 Socket socket = serverSocket.accept();  
 new Thread(new Server(socket)).start();  
 }  
 }  
 }  
}

Также создан класс Server, который занимается обработкой запросов: добавление нового пользователя, отправка списка всех пользователей клиенту, отправка объектов другого пользователя клиенту, удаление пользователя из общего списка при отключении.

public class Server implements Runnable {  
 private Socket socket;  
 private static ArrayList<String> *users* = new ArrayList<>();  
 private static Map<String, Vector<House>> *houses* = new LinkedHashMap<>();  
  
 private static final String *SET\_USER* = "setUser";  
 private static final String *GET\_USERS* = "getUsers";  
 private static final String *GET\_OBJECTS* = "getObjects";  
 private static final String *DISCONNECT* = "disconnect";  
  
 public Server(Socket socket) { this.socket = socket; }  
  
 @Override  
 public void run() {  
 try {  
 ObjectInputStream objectInputStream = new ObjectInputStream(socket.getInputStream());  
 ObjectOutputStream objectOutputStream = new ObjectOutputStream(socket.getOutputStream());  
  
 String user = objectInputStream.readUTF();  
 String command = objectInputStream.readUTF();  
 String target = objectInputStream.readUTF();  
 System.*out*.println(user);  
 System.*out*.println(command);  
 System.*out*.println(target);  
  
 switch (command) {  
 case *SET\_USER* -> *users*.add(user);  
 case *GET\_USERS* -> {  
 objectOutputStream.writeObject(*users*);  
 *houses*.put(user, (Vector<House>) objectInputStream.readObject());  
 }  
 case *GET\_OBJECTS* -> objectOutputStream.writeObject(*houses*.get(target));  
 case *DISCONNECT* -> {  
 *users*.remove(user);  
 *houses*.remove(user);  
 }  
 default -> objectOutputStream.writeObject("ERROR");  
 }  
  
 } catch (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
}







*Рис.1-3 – Скриншоты работы программы*

**Вывод:**

В ходе лабораторной работы были изучены особенности реализации сетевых приложений в Java. Был создан TCP-сервер, ожидающий подключения пользователя и обрабатывающий его запросы, а так-же клиент, высылающий запросы серверу.

**Программный код:**

https://github.com/AVT-808/AVT-808/tree/Popov-Dmitry-7