МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

**Лабораторная работа №6**

по дисциплине «Технология программирования»

на тему: «Сетевые «клиент-серверные» приложения»

|  |  |
| --- | --- |
| Группа: АВТ-808 | Преподаватель: Михайленко Дмитрий Анатольевич |
| Студент: Кузьмина Александра |  |
| Вариант: 1 |  |
|  |  |

Новосибирск, 2020

**Цель работы**

Изучить особенности реализации сетевых приложений в Java.

**Задание**

Доработать программу, созданную в лабораторной работу № 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 1**

Реализовать выбор из одного подключенного клиента N случайных объектов и поместить в текущую симуляцию.

**Ход работы**

В ходе решения поставленных задач доработки программы были добавлены новые классы и внесены соответствующие изменения в ранее разработанные:

1. Создан класс **Client**, отвечающий за все взаимодействие с сервером. Он высылают запросы на получение объектов другого пользователя, получение списка пользователей, добавление/исключение пользователя из общего списка.



2. Создан класс **Server**, представляющий собой сервер. Он занимается обработкой полученных от клиентов запросов.



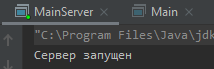
3. Создан класс **MainServer**, который запускает сервер, ожидает подключения и для каждого подключенного клиента создает свой поток.



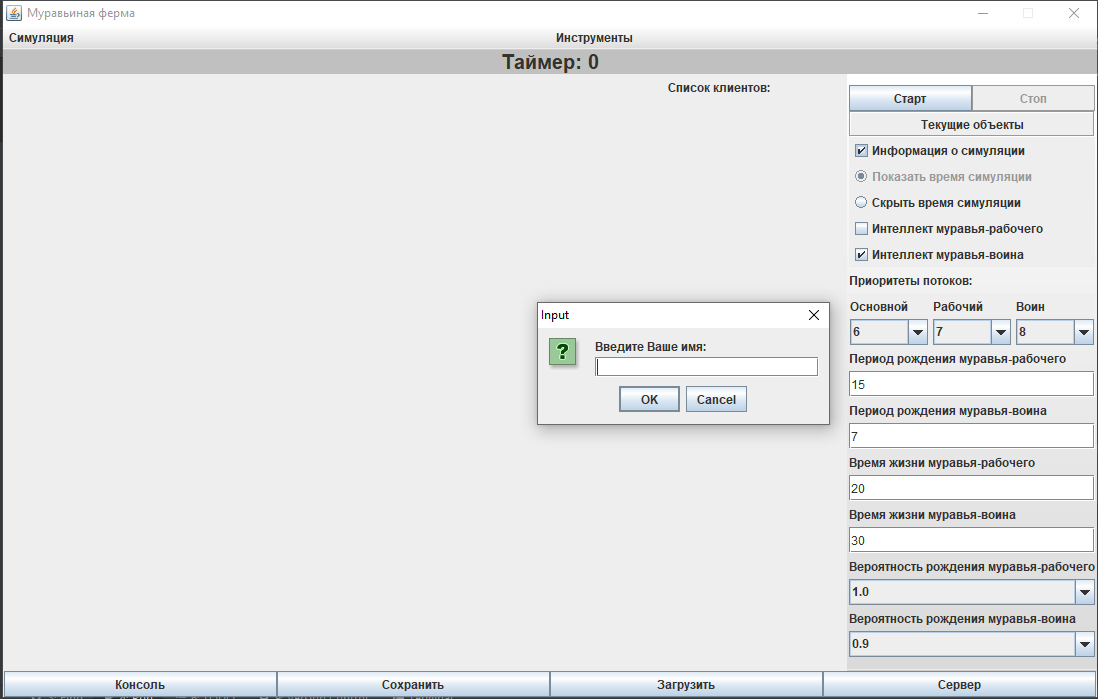
Остальные классы программы были переработаны в соответствии с новыми задачами.

**Скриншоты работы программы**

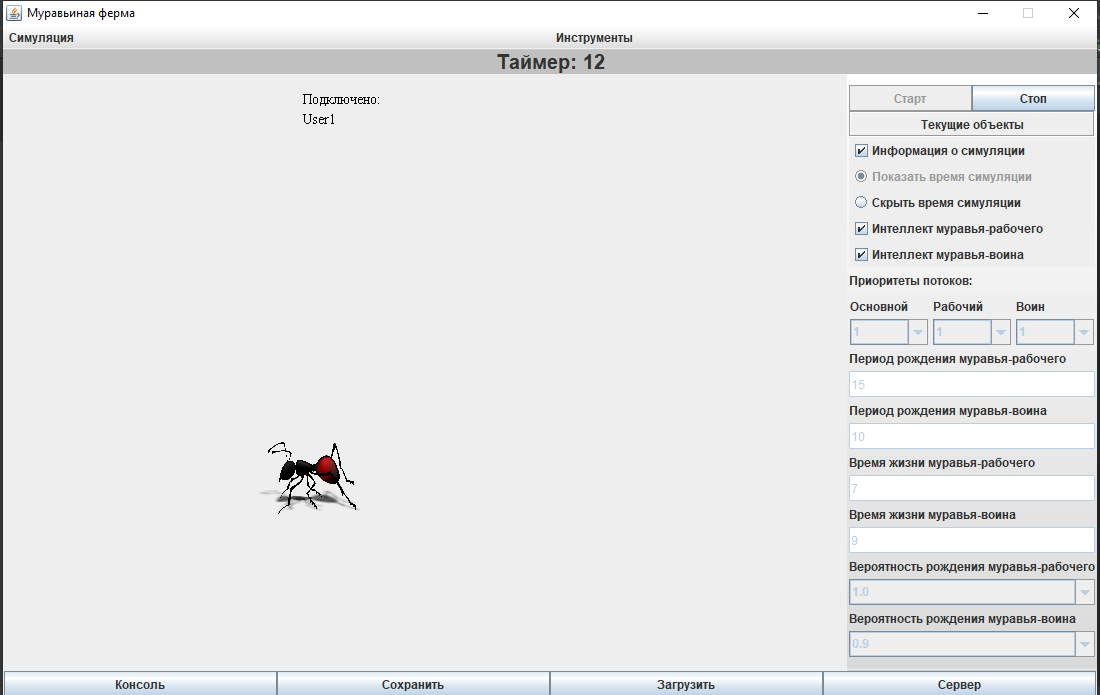
Запуск сервера:



Окно до запуска симуляции:



Окно во время симуляции:



**Вывод**

При выполнении данной работы были изучены особенности сетевых приложений в Java. В результате было создано отдельное консольное приложение, которое играет роль TCP-сервера. Клиентом стало приложение, что было создано в предыдущих работах. В области симуляции реализовано отображение списка всех подключенных к серверу клиентов. Также было реализовано специальное взаимодействие по TCP с другими клиентами через сервер

**Листинг программы**

**Server**

package AntFarm;  
  
import java.io.IOException;  
import java.io.ObjectInputStream;  
import java.io.ObjectOutputStream;  
import java.net.Socket;  
import java.util.LinkedHashMap;  
import java.util.Map;  
import java.util.Vector;  
  
public class Server extends Thread {  
 private Socket socket;  
 private static Vector<String> *users* = new Vector<>();  
 private static Map<String, Vector<Ant>> *ants* = new LinkedHashMap<>();  
 private static String *setUser* = "Добавить пользователя";  
 private static String *getUsers* = "Получить список пользователей";  
 private static String *getObjects* = "Передать объекты";  
 private static String *disconnect* = "Отключиться";  
  
 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);  
 if (command.equals(*setUser*))  
 {  
 *users*.add(user);  
 }  
 else  
 {  
 if (command.equals(*getUsers*))  
 {  
 objectOutputStream.writeObject(*users*);  
 *ants*.put(user, (Vector<Ant>) objectInputStream.readObject());  
 }  
 else  
 {  
 if (command.equals(*getObjects*))  
 {  
 objectOutputStream.writeObject(*ants*.get(target));  
 }  
 else {  
 if (command.equals(*disconnect*))  
 {  
 *users*.remove(user);  
 *ants*.remove(user);  
 }  
 else {  
 objectOutputStream.writeObject("Ошибка!");  
 }  
 }  
 }  
 }  
 }  
 catch (IOException | ClassNotFoundException e)  
 {  
 e.printStackTrace();  
 }  
 }  
}

**Client**

package AntFarm;  
  
import javax.imageio.ImageIO;  
import javax.swing.\*;  
import java.io.IOException;  
import java.io.ObjectInputStream;  
import java.io.ObjectOutputStream;  
import java.net.Socket;  
import java.util.Vector;  
  
public class Client  
{  
 private static String *username*;  
 private static Socket *socket*;  
 private static String *host* = "localhost";  
 private static int *port* = 3333;  
 private static String *setUser* = "Добавить пользователя";  
 private static String *getUsers* = "Получить список пользователей";  
 private static String *getObjects* = "Передать объекты";  
 private static String *disconnect* = "Отключиться";  
 private static Vector<String> *users* = new Vector<>();  
  
 Client() {  
 *username* = JOptionPane.*showInputDialog*("Введите Ваше имя: ");  
 try  
 {  
 *socket* = new Socket(*host*, *port*);  
 ObjectOutputStream objectOutputStream = new ObjectOutputStream(*socket*.getOutputStream());  
 objectOutputStream.writeUTF(*username*);  
 objectOutputStream.writeUTF(*setUser*);  
 objectOutputStream.writeUTF("Пользователь для передачи объектов");  
 objectOutputStream.writeObject(null);  
 objectOutputStream.flush();  
 }  
 catch (IOException e)  
 {  
 e.printStackTrace();  
 }  
 }  
  
 public static Vector<String> getUsers() {  
 try  
 {  
 *socket* = new Socket(*host*, *port*);  
 ObjectOutputStream objectOutputStream = new ObjectOutputStream(*socket*.getOutputStream());  
 ObjectInputStream objectInputStream = new ObjectInputStream(*socket*.getInputStream());  
 objectOutputStream.writeUTF(*username*);  
 objectOutputStream.writeUTF(*getUsers*);  
 objectOutputStream.writeUTF("Пользователь для передачи объектов");  
 objectOutputStream.writeObject(Singleton.*getAnts*());  
 objectOutputStream.flush();  
 *users* = (Vector<String>) objectInputStream.readObject();  
 objectInputStream.close();  
 }  
 catch (IOException | ClassNotFoundException e)  
 {  
 e.printStackTrace();  
 }  
 return *users*;  
 }  
  
 public static void getAnts(Habitat antFarm) {  
 int N = 3;  
 int workers = 0, warriors = 0;  
 String targetUser = "";  
 try  
 {  
 while (!*users*.contains(targetUser))  
 {  
 targetUser = JOptionPane.*showInputDialog*("Введите имя пользователя для получения объектов:");  
 }  
 *socket* = new Socket(*host*,*port*);  
 ObjectOutputStream objectOutputStream = new ObjectOutputStream(*socket*.getOutputStream());  
 ObjectInputStream objectInputStream = new ObjectInputStream(*socket*.getInputStream());  
 objectOutputStream.writeUTF(*username*);  
 objectOutputStream.writeUTF(*getObjects*);  
 objectOutputStream.writeUTF(targetUser);  
 objectOutputStream.writeObject(Singleton.*getAnts*());  
 System.*out*.println(targetUser);  
 objectOutputStream.flush();  
 Vector<Ant> ants = (Vector<Ant>) objectInputStream.readObject();  
 System.*out*.println("Изначальный размер массива: " + Singleton.*getArraySize*());  
 System.*out*.println("Полученный вектор: " + ants + "\n");  
 if (ants.size() < N)  
 N = ants.size();  
 for (int i = 0; i < N;i++)  
 {  
 Singleton.*addAnt*(ants.get(i), ants.get(i).id, antFarm.getTime());  
 if (ants.get(i).getClass() == AntWorker.class)  
 {  
 Singleton.*getAnts*().get(i).setImage(ImageIO.*read*(Client.class.getResource("/AntWorker.png")));  
 workers++;  
 }  
 else  
 {  
 Singleton.*getAnts*().get(i).setImage(ImageIO.*read*(Client.class.getResource("/AntWarrior.png")));  
 warriors++;  
 }  
 }  
 System.*out*.println("Новый размер массива: " + Singleton.*getArraySize*());  
 System.*out*.println("Singleton: " + Singleton.*getAnts*());  
 antFarm.setAntsAmount(Singleton.*getArraySize*());  
 antFarm.setWorkersAmount(antFarm.getWorkersAmount() + workers);  
 antFarm.setWarriorsAmount(antFarm.getWarriorsAmount() + warriors);  
 objectInputStream.close();  
 }  
 catch (IOException | ClassNotFoundException e)  
 {  
 e.printStackTrace();  
 }  
 }  
  
 public static void disconnect() {  
 try  
 {  
 *socket* = new Socket(*host*,*port*);  
 ObjectOutputStream objectOutputStream = new ObjectOutputStream(*socket*.getOutputStream());  
 objectOutputStream.writeUTF(*username*);  
 objectOutputStream.writeUTF(*disconnect*);  
 objectOutputStream.writeUTF("Пользователь для передачи объектов");  
 objectOutputStream.writeObject(null);  
 objectOutputStream.flush();  
 }  
 catch (IOException e)  
 {  
 e.printStackTrace();  
 }  
 }  
}

**MainServer**

package AntFarm;  
  
import java.io.IOException;  
import java.net.ServerSocket;  
import java.net.Socket;  
  
public class MainServer  
{  
 public static void main(String[] args)  
 {  
 try  
 {  
 System.*out*.println("Сервер запущен");  
 int port = 3333;  
 ServerSocket ss = new ServerSocket(port);  
 while (true)  
 {  
 Socket s = ss.accept();  
 Server p = new Server(s);  
 p.start();  
 }  
 }  
 catch(IOException e)  
 {  
 e.printStackTrace();  
 }  
 }  
}