**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ “НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”**

Лабораторная работа №6

по дисциплине «Технологии и методы программирования»

Сетевые «клиент-серверные» приложения

Группа: **АВТ-042**

Студенты:

**Антонов А.Р.**

Преподаватель: **Булатов А.Д**

Новосибирск 2022

1. **Цель**

Изучить особенности реализации сетевых приложений в Kotlin.

1. **Задание**

Доработать программу, созданную в лабораторной работе № 4:

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.

**Вариант 10**

Реализовать возможность синхронной установки вероятностей появления физических и юридических лиц на текущем и одном из подключенных клиентов, т.е. при установке этих параметров на выбранном клиенте должны выставиться такие же параметры.

1. **UML-диаграмма**

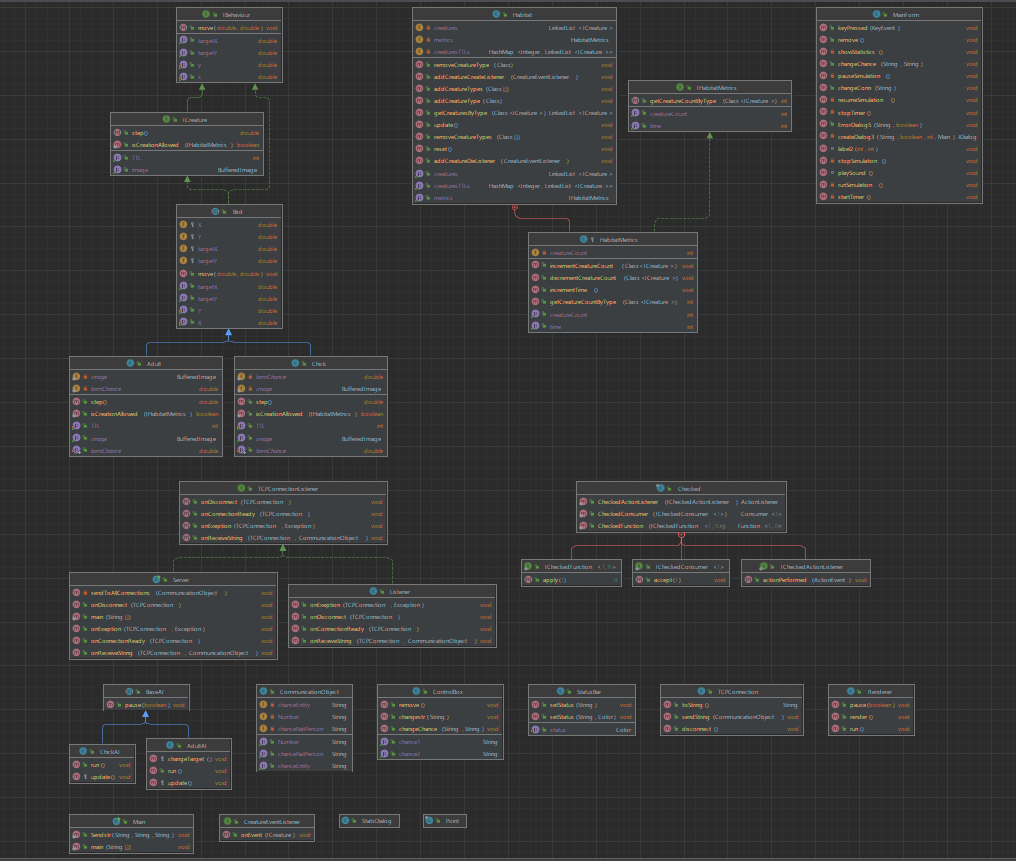


Рисунок 1 UML-диаграмма

1. **Пример работы программы**

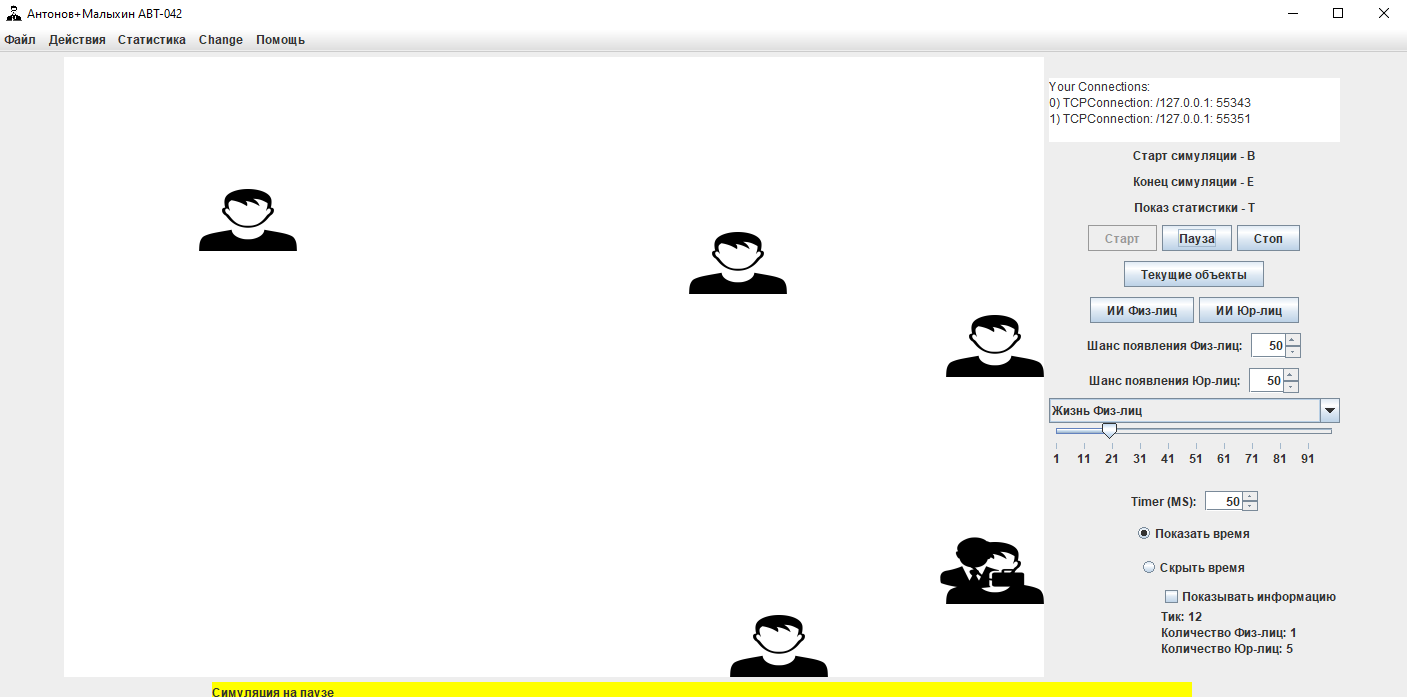


Рисунок 2 Пример работы программы

1. **Вывод**

Мы изучили особенности реализации сетевых приложений в Java. Была доработана программа, созданная в лабораторной работе № 4. Были выполнены все требования согласно варианту.

**Приложение**

**class Listener**

package lab;  
  
import Network.CommunicationObject;  
import Network.TCPConnection;  
import Network.TCPConnectionListener;  
import lab.frames.MainForm;  
  
import java.util.ArrayList;  
  
public class Listener implements TCPConnectionListener {  
 private final MainForm g;  
  
 public Listener(MainForm a){  
 g=a;  
 }  
  
 @Override  
 public void onConnectionReady(TCPConnection tcpConnection) {  
 System.*out*.println("Connection is ready....");  
  
 }  
  
 @Override  
 public void onReceiveString(TCPConnection tcpConnection, CommunicationObject args) throws Exception {  
 try{  
 if(Integer.*parseInt*(args.getNumber())==-1){  
 g.remove();  
 return;  
 }  
 if(Integer.*parseInt*(args.getNumber())==-2){  
 g.ErrorDialog5("Wrong Client!!!",false);  
 return;  
 }  
 g.changeChance(args.getChanceEntity(), args.getChanceNatPerson());}  
 catch(NumberFormatException e){  
 g.changeConn(args.getNumber());  
 }  
 }  
  
 @Override  
 public void onDisconnect(TCPConnection tcpConnection) {  
 g.ErrorDialog5("Disconnect",true);  
 System.*out*.println("Finish");  
 }  
  
 @Override  
 public void onExeption(TCPConnection tcpConnection, Exception e) {  
 g.ErrorDialog5(e.toString(),true);  
 }  
}

**class Server**

package Server;  
  
import Network.CommunicationObject;  
import Network.TCPConnection;  
import Network.TCPConnectionListener;  
  
import java.io.\*;  
import java.net.ServerSocket;  
import java.util.ArrayList;  
  
  
  
public class Server implements TCPConnectionListener {  
  
 public static void main(String [] arg) {  
 new Server();  
 }  
 private final ArrayList<TCPConnection>connections= new ArrayList<>();  
 private Server(){  
 System.*out*.println("Server running...");  
 try(ServerSocket serverSocket=new ServerSocket(8000)){  
 while (true){  
 try{  
 TCPConnection aa= new TCPConnection(this, serverSocket.accept());  
 }catch(IOException e){  
 System.*out*.println("TCPConnection exception"+e);  
 }  
 }  
 }catch(IOException e){  
 throw new RuntimeException(e);  
 }  
 }  
  
  
 @Override  
 public synchronized void onConnectionReady(TCPConnection tcpConnection) {  
 connections.add(tcpConnection);  
 System.*out*.println("New Connection Received...");  
 if(connections.size()==1)connections.get(0).sendString(new CommunicationObject("0) "+connections.get(0).toString()));  
 else {  
 for (int i = 0; i < connections.size() - 1; i++)  
 connections.get(i).sendString(new CommunicationObject(connections.size() - 1 + ") " + connections.get(connections.size() - 1).toString()));  
 for(int i = 0; i < connections.size(); i++)  
 connections.get(connections.size() - 1).sendString(new CommunicationObject(i + ") " + connections.get(i).toString()));  
 }  
 }  
  
 @Override  
 public synchronized void onReceiveString(TCPConnection tcpConnection, CommunicationObject gag) throws IOException {  
 try{  
 if(Integer.*parseInt*(gag.getNumber())>connections.size()-1||Integer.*parseInt*(gag.getNumber())<0)tcpConnection.sendString(new CommunicationObject("-2"));  
 else connections.get(Integer.*parseInt*(gag.getNumber())).sendString(new CommunicationObject(gag.getChanceEntity(), gag.getChanceNatPerson(), gag.getNumber()));}  
 catch(Exception e){tcpConnection.sendString(new CommunicationObject("-2"));}  
 }  
  
  
  
 @Override  
 public synchronized void onDisconnect(TCPConnection tcpConnection) {  
 connections.remove(tcpConnection);  
 sendToAllConnections(new CommunicationObject("-1"));  
 for(int i=0;i<connections.size();i++)  
 sendToAllConnections(new CommunicationObject(i+") "+connections.get(i).toString()));  
 }  
  
 @Override  
 public synchronized void onExeption(TCPConnection tcpConnection, Exception e) {  
 System.*out*.println("TCPConnection exception: "+e);  
  
 }  
  
 private void sendToAllConnections(CommunicationObject value){  
 for(int i=0;i< connections.size();i++){  
 connections.get(i).sendString(value);  
 }  
 }  
}

**interface TCPConnectionListener**

package Network;  
import java.util.ArrayList;  
  
public interface TCPConnectionListener {  
  
 void onConnectionReady(TCPConnection tcpConnection);  
 void onReceiveString(TCPConnection tcpConnection,CommunicationObject obj) throws Exception;  
 void onDisconnect(TCPConnection tcpConnection);  
 void onExeption(TCPConnection tcpConnection, Exception e);  
  
}

**interface TCPConnectionListener**

package Network;  
  
import java.io.Serializable;  
  
public class CommunicationObject implements Serializable {  
 private String chanceEntity;  
 private String chanceNatPerson;  
 private String Number;  
  
  
 public CommunicationObject(String chance1, String chance2, String chance3){  
 chanceEntity=chance1;  
 chanceNatPerson=chance2;  
 Number=chance3;  
 }  
 public CommunicationObject(String conn){  
 Number=conn;  
 }  
 public String getChanceEntity() {  
 return chanceEntity;  
 }  
 public String getChanceNatPerson() {  
 return chanceNatPerson;  
 }  
 public String getNumber() {  
 return Number;  
 }  
}

**class TCPConnection**

package Network;  
  
import java.io.\*;  
import java.net.Socket;  
import java.util.ArrayList;  
  
public class TCPConnection {  
 private final Socket socket;  
 private final Thread rxThread;  
 private final TCPConnectionListener eventListener;  
 private final ObjectOutputStream out1;  
 private final ObjectInputStream in1;  
  
 public TCPConnection(TCPConnectionListener eventListener, String ipADDr, int port) throws IOException{  
 this(eventListener, new Socket(ipADDr, port));  
 }  
  
 public TCPConnection(TCPConnectionListener eventListener, Socket socket) throws IOException{  
 this.eventListener=eventListener;  
 this.socket=socket;  
 out1=new ObjectOutputStream(socket.getOutputStream());  
 in1=new ObjectInputStream(socket.getInputStream());  
 rxThread=new Thread(new Runnable() {  
 @Override  
 public void run() {  
 try{  
 eventListener.onConnectionReady(TCPConnection.this);  
 while (!rxThread.isInterrupted()){  
 eventListener.onReceiveString(TCPConnection.this, (CommunicationObject) in1.readObject());  
 }  
 } catch (IOException e){  
 eventListener.onExeption(TCPConnection.this, e);  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally{  
 eventListener.onDisconnect(TCPConnection.this);  
 }  
 }  
 });  
 rxThread.start();  
 }  
 public synchronized void sendString(CommunicationObject objs){  
 try {  
 out1.writeObject(objs);  
 out1.flush();  
 } catch(IOException e){  
 eventListener.onExeption(TCPConnection.this, e);  
 disconnect();  
 }  
 }  
 public synchronized void disconnect(){  
 rxThread.interrupt();  
 try{  
 socket.close();  
 }catch(IOException e){  
 eventListener.onExeption(TCPConnection.this, e);  
 }  
  
 }  
 @Override  
 public String toString(){  
 return "TCPConnection: "+socket.getInetAddress()+": "+socket.getPort();  
 }  
}