МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

НГТУ

Лабораторная работа №6

по предмету «Технология программирования»

на тему «Сетевые «клиент-серверные» приложения»

Факультет: АВТ Преподаватель:

Группа: АВТ-809 Михайленко Д.А.

Студент: Швыдко Е.А.

Новосибирск 2020

Задание:

**Вариант 11**

Реализовать возможность обмена объектами с выбранным клиентом. У одного собираются все кошки, у другого все собаки.

Цели работы:

1. Изучить особенности реализации сетевых приложений в Java.
2. Доработать программу, созданную в лабораторной работе № 5:
3. создать отдельное консольное приложение, которое будет играть роль TCP-сервера. Клиентом будет приложение, которое было создано в предыдущих работах;
4. 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;
5. запрограммировать специальное взаимодействие по TCP с другими клиентами через сервер по варианту;

Ход разработки:

* Был разработано приложение Server, реализующее работу сервера

**public** **class** Server {

**public** **static** **final** **String** GET\_CLIENTS = "get clients";

**public** **static** **final** **String** SWAP\_WITH\_CLIENT = "connect client";

**public** **static** **final** **String** SEND\_OBJ = "sendObj";

**public** **static** **final** **String** DISCONNECT = "disconnect";

**static** **TreeMap**<**Integer**, ConnectionThread> clients = **new** **TreeMap**<>();

**static** **List**<**Integer**> ids = **new** **ArrayList**<>();

**private** **static** int id = 0;

**public** **static** void main(**String**[] args) {

**try** {

**System**.out.println("Server is running");

int port = 1337;

**ServerSocket** serverSocket = **new** **ServerSocket**(port);

// Ждет клиентов и для каждого создает отдельный поток

**while** (**true**) {

**Socket** s = serverSocket.accept();

ConnectionThread connectionThread = **new** ConnectionThread(s, id);

clients.put(id, connectionThread);

ids.add(id);

id++;

connectionThread.start();

update();

}

} **catch** (**Exception** e) {

e.printStackTrace();

}

}

**static** void update(){

**for** (int i = 0; i < ids.size()-1; i++) {

Server.clients.get(ids.get(i)).updateClients();

}

}

**static** void updateAll(){

**for** (int i = 0; i < ids.size(); i++) {

Server.clients.get(ids.get(i)).updateClients();

}

}

}

* Был реализован класс Client, реализующий работу клиентского сокета

**public** **class** Client **extends** Thread {

**private** **Socket** socket;

**private** **ObjectOutputStream** outStream;

**private** **ObjectInputStream** inStream;

**private** boolean connected = **true**;

**private** **String** command;

**private** SwapDialog owner;

**private** int targetId;

**private** int id;

**private** **Vector**<Animal> toArrive = **new** **Vector**<>(), toDepart = **new** **Vector**<>();

Client(SwapDialog owner) {

**this**.owner = owner;

}

**@Override**

**public** void run() {

**try** {

**String** host = "localhost";

int port = 1337;

socket = **new** **Socket**(host, port);

connected = **true**;

outStream = **new** **ObjectOutputStream**(socket.getOutputStream());

inStream = **new** **ObjectInputStream**(socket.getInputStream());

**List**<**Integer**> ids = (**List**<**Integer**>) inStream.readObject();

id = inStream.readInt();

owner.showList(ids, id);

**System**.out.println("List received");

**while** (connected) {

**if** (inStream.available() == 0) {

**if** (command != **null**) {

**switch** (command) {

**case** "swap":

outStream.writeUTF(Server.SWAP\_WITH\_CLIENT);

outStream.writeInt(targetId);

outStream.writeObject(Singleton.getVector().getArrayForSend());

outStream.flush();

command = **null**;

**break**;

**case** "disconnect":

**System**.out.println("Somewho disconnected");

outStream.writeUTF(Server.DISCONNECT);

outStream.flush();

command = **null**;

**break**;

**default**:

**break**;

}

}

} **else** {

**String** serverCommand = inStream.readUTF();

**System**.out.println(serverCommand);

int senderID;

**switch** (serverCommand) {

**case** "swapObj":

toArrive = (**Vector**<Animal>) inStream.readObject();

Singleton.getVector().setArrayToArrive(toArrive);

**System**.out.println("Размер полученного " + toArrive.size());

**if**(!toArrive.isEmpty()){

**if**(toArrive.get(0) **instanceof** Cat) {

Singleton.getVector().createArrayToSend(**false**);

Singleton.getVector().replaceTypeOfObject();

}

**else** **if**(toArrive.get(0) **instanceof** Dog) {

Singleton.getVector().createArrayToSend(**true**);

Singleton.getVector().replaceTypeOfObject();

}

}

Singleton.getVector().getHabitat().repaint();

senderID = inStream.readInt();

toDepart = Singleton.getVector().getArrayForSend();

outStream.writeUTF(Server.SEND\_OBJ);

outStream.writeObject(toDepart);

outStream.writeInt(senderID);

outStream.flush();

**break**;

**case** "getObj":

toArrive = (**Vector**<Animal>) inStream.readObject();

Singleton.getVector().setArrayToArrive(toArrive);

**System**.out.println("Размер полученного " + toArrive.size());

**if**(!toArrive.isEmpty()){

**if**(toArrive.get(0) **instanceof** Cat) Singleton.getVector().replaceTypeOfObject();

**else** **if**(toArrive.get(0) **instanceof** Dog) Singleton.getVector().replaceTypeOfObject();

}

Singleton.getVector().getHabitat().repaint();

**break**;

**case** "updateClients":

ids = (**List**<**Integer**>) inStream.readObject();

owner.showList(ids, id);

**break**;

}

}

}

} **catch** (**Exception** e) {

e.printStackTrace();

} **finally** {

closeAll();

}

}

**public** void swapWithClient(int id) {

command = "swap";

targetId = id;

}

**public** void disconnect() {

command = "disconnect";

}

**private** void closeAll() {

**System**.out.println("closeAll");

connected = **false**;

**try** {

inStream.close();

outStream.close();

socket.close();

} **catch** (**IOException** e) {

e.printStackTrace();

}

}

}

* Был реализован класс ConnectionThread, реализующий функции сетевой «нити», по которой передаются объекты

**public** **class** ConnectionThread **extends** Thread {

**private** **ObjectOutputStream** outStream;

**private** **ObjectInputStream** inStream;

**private** **Socket** socket;

**private** int id;

**private** boolean connected = **true**;

ConnectionThread(**Socket** socket, int id) {

**this**.socket = socket;

**this**.id = id;

}

**@Override**

**public** void run() {

**try** {

outStream = **new** **ObjectOutputStream**(socket.getOutputStream());

inStream = **new** **ObjectInputStream**(socket.getInputStream());

outStream.writeObject(Server.ids);

outStream.writeInt(id);

outStream.flush();

**while** (connected) {

**System**.out.println("read");

**String** command = inStream.readUTF();

**System**.out.println(command);

**switch** (command) {

**case** Server.SWAP\_WITH\_CLIENT:

int targetID = inStream.readInt();

**Vector**<Animal> obj = (**Vector**<Animal>) inStream.readObject();

Server.clients.get(targetID).outStream.writeUTF("swapObj");

Server.clients.get(targetID).outStream.writeObject(obj);

Server.clients.get(targetID).outStream.writeInt(id);

Server.clients.get(targetID).outStream.flush();

**break**;

**case** Server.SEND\_OBJ:

**Vector**<Animal> objs = (**Vector**<Animal>) inStream.readObject();

int senderID = inStream.readInt();

Server.clients.get(senderID).outStream.writeUTF("getObj");

Server.clients.get(senderID).outStream.writeObject(objs);

Server.clients.get(senderID).outStream.flush();

**break**;

**case** Server.DISCONNECT:

connected = **false**;

disconnect();

**break**;

**default**:

**System**.err.println("Unknown command");

}

}

} **catch** (**Exception** e) {

e.printStackTrace();

} **finally** {

disconnect();

Server.updateAll();

}

}

**private** void disconnect() {

**try** {

**System**.out.println("disconnect");

**if** (inStream != **null**) inStream.close();

**if** (outStream != **null**) outStream.close();

socket.close();

connected = **false**;

Server.clients.remove(id);

**for** (int i = 0; i < Server.ids.size(); i++)

**if** (Server.ids.get(i) == id)

Server.ids.remove(i);

} **catch** (**Exception** e) {

e.printStackTrace();

}

}

**public** void updateClients() {

**try** {

outStream.writeUTF("updateClients");

**List**<**Integer**> isds = **new** **ArrayList**<>();

**for** (int i = 0; i < Server.ids.size(); i++)

isds.add(Server.ids.get(i));

outStream.writeObject(isds);

outStream.flush();

} **catch** (**IOException** e) {

e.printStackTrace();

}

}

}

* Был разработан класс «SwapDialog», позволяющий открыть доступ к обмену с пользователями:

**public** **class** SwapDialog **extends** JDialog {

**private** JList<**Object**> clients;

**private** **List**<**Integer**> clientList;

**private** **String**[] items;

**private** Client client;

**private** JRadioButton radioCat;

SwapDialog(MFrame parent) {

**super**(parent, "Client-Server", **false**);

setSize(288,349);

setLayout(**new** FlowLayout());

setResizable(**false**);

setLocationRelativeTo(parent);

JPanel swapDialogPanel = **new** JPanel(**new** FlowLayout());

clients = **new** JList<>();

JScrollPane scrollPane = **new** JScrollPane(clients);

swapDialogPanel.add(scrollPane);

scrollPane.setPreferredSize(**new** Dimension(270,250));

//Buttons

JButton connectButton = **new** JButton("Connect");

JButton disconnectButton = **new** JButton("Disconnect");

JButton swapButton = **new** JButton("Swap");

swapDialogPanel.add(connectButton);

swapDialogPanel.add(disconnectButton);

swapDialogPanel.add(swapButton);

radioCat = **new** JRadioButton("Cat");

JRadioButton radioDog = **new** JRadioButton("Dog");

ButtonGroup radioButtons = **new** ButtonGroup();

radioButtons.add(radioCat);

radioButtons.add(radioDog);

radioButtons.setSelected(radioCat.getModel(),**true**);

swapDialogPanel.add(radioCat);

swapDialogPanel.add(radioDog);

//Area

//Listeners

connectButton.addActionListener(**new** ActionListener() {

**@Override**

**public** void actionPerformed(ActionEvent e) {

connect();

}

});

disconnectButton.addActionListener(**new** ActionListener() {

**@Override**

**public** void actionPerformed(ActionEvent e) {

disconnect();

}

});

swapButton.addActionListener(**new** ActionListener() {

**@Override**

**public** void actionPerformed(ActionEvent e) {

Singleton.getVector().createArrayToSend(radioCat.isSelected());

swap();

}

});

//Singleton.getCollection().createArrayToSend(true);

setContentPane(swapDialogPanel);

setVisible(**true**);

}

**private** void connect() {

client = **new** Client(**this**);

client.start();

}

**private** void disconnect() {

client.disconnect();

clients.setListData(**new** **String**[0]);

}

**private** void swap() {

**if** (clientList == **null** || clientList.isEmpty()) **return**;

**if** (clients.getSelectedIndex() == -1) **return**;

client.swapWithClient(clientList.get(clients.getSelectedIndex()));

**try** {

**Thread**.sleep(500);

//Singleton.getCollection().replaceTypeOfObject(radioAuto.isSelected());

} **catch** (**InterruptedException** e) {

e.printStackTrace();

}

}

**public** void showList(**List**<**Integer**> ids, int id) {

clientList = ids;

items = **null**;

items = **new** **String**[ids.size()];

**for** (int i = 0; i < ids.size(); i++) {

**if** (ids.get(i) == id){

items[i] = "Клиент №" + ids.get(i) + " <- Это вы";

}**else**{

items[i] = "Клиент №" + ids.get(i);

}

}

clients.setListData(items);

}

}

**Выводы:**

В результате выполнения работы были изучены особенности реализации сетевых приложений. Была разработана сетевая «клиент-серверная» программа, позволяющая клиентам собрать у себя объекты одного типа с другого клиента.