|  |
| --- |
|  |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** |
| Институт информационных технологий (ИТ) |
| Кафедра … |

|  |  |
| --- | --- |
| **ОТЧЕТ**  **ПО ЛАБОРАТОРНОЙ РАБОТЕ №3** | |
| **по дисциплине** |  |
| **«Программирование на языке Java»**  **Тема: Программирование JAVA сокетов** | |
| Выполнил студент группы ИКБО-20-19 Анваржонов Ж.Т |  |
| Принял ассистент кафедры ИиППО | Степанов П.В |

|  |  |  |
| --- | --- | --- |
| Практические работы выполнены | « 10 »октября 2020г. |  |
| «Зачтено» | « » 2020 г. |  |

Москва 2020

**Теоретическое введение**

**Цель работы:** знакомство студентов с одной из базовых технологии реализации конечной точки для передачи и приема данных по сети - сокетом.

**Постановка задачи**

**Программный код**

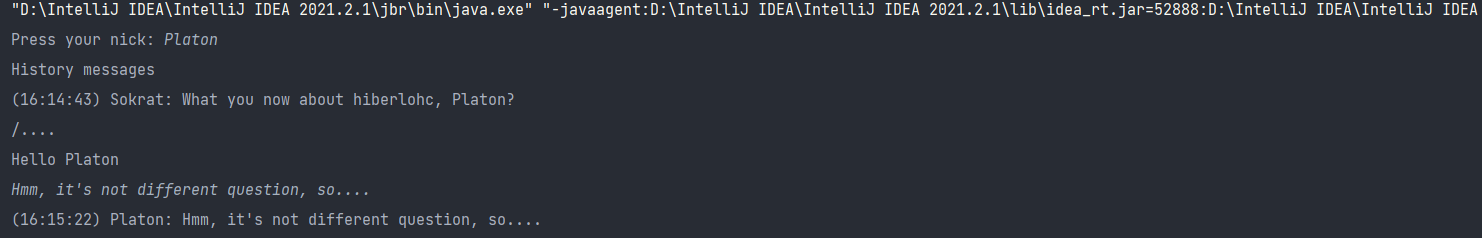
**Server.java:**

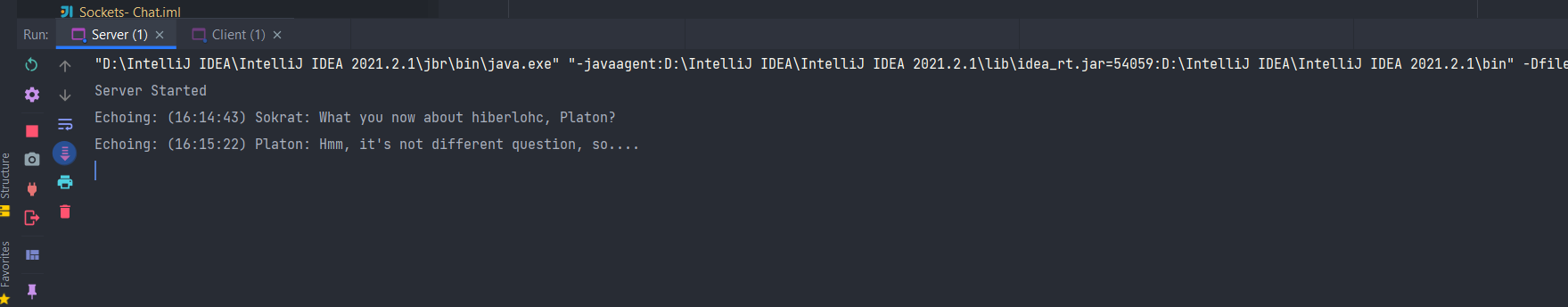
*import* java.io.\*;  
*import* java.net.\*;  
*import* java.util.LinkedList;  
  
*/\*\*  
 \* проект реализует консольный многопользовательский чат.  
 \* вход в программу запуска сервера - в классе Server.  
 \** ***@author*** *izotopraspadov, the tech  
 \** ***@version*** *2.0  
 \*/  
  
class* ServerSomthing *extends* Thread {  
  
 *private* Socket socket; *// сокет, через который сервер общается с клиентом,  
 // кроме него - клиент и сервер никак не связаны  
 private* BufferedReader in; *// поток чтения из сокета  
 private* BufferedWriter out; *// поток завписи в сокет  
  
 /\*\*  
 \* для общения с клиентом необходим сокет (адресные данные)  
 \** ***@param socket*** *\** ***@throws*** *IOException  
 \*/  
  
 public* ServerSomthing(Socket socket) *throws* IOException {  
 *this*.socket = socket;  
 *// если потоку ввода/вывода приведут к генерированию искдючения, оно проброситься дальше* in = *new* BufferedReader(*new* InputStreamReader(socket.getInputStream()));  
 out = *new* BufferedWriter(*new* OutputStreamWriter(socket.getOutputStream()));  
 Server.*story*.printStory(out); *// поток вывода передаётся для передачи истории последних 10  
 // сооюбщений новому поключению* start(); *// вызываем run()* }  
 @Override  
 *public void* run() {  
 String word;  
 *try* {  
 *// первое сообщение отправленное сюда - это никнейм* word = in.readLine();  
 *try* {  
 out.write(word + "\n");  
 out.flush(); *// flush() нужен для выталкивания оставшихся данных  
 // если такие есть, и очистки потока для дьнейших нужд* } *catch* (IOException ignored) {}  
 *try* {  
 *while* (*true*) {  
 word = in.readLine();  
 *if*(word.equals("stop")) {  
 *this*.downService(); *// харакири  
 break*; *// если пришла пустая строка - выходим из цикла прослушки* }  
 System.out.println("Echoing: " + word);  
 Server.*story*.addStoryEl(word);  
 *for* (ServerSomthing vr : Server.*serverList*) {  
 vr.send(word); *// отослать принятое сообщение с привязанного клиента всем остальным влючая его* }  
 }  
 } *catch* (NullPointerException ignored) {}  
  
  
 } *catch* (IOException e) {  
 *this*.downService();  
 }  
 }  
  
 */\*\*  
 \* отсылка одного сообщения клиенту по указанному потоку  
 \** ***@param msg*** *\*/  
 private void* send(String msg) {  
 *try* {  
 out.write(msg + "\n");  
 out.flush();  
 } *catch* (IOException ignored) {}  
  
 }  
  
 */\*\*  
 \* закрытие сервера  
 \* прерывание себя как нити и удаление из списка нитей  
 \*/  
 private void* downService() {  
 *try* {  
 *if*(!socket.isClosed()) {  
 socket.close();  
 in.close();  
 out.close();  
 *for* (ServerSomthing vr : Server.*serverList*) {  
 *if*(vr.equals(*this*)) vr.interrupt();  
 Server.*serverList*.remove(*this*);  
 }  
 }  
 } *catch* (IOException ignored) {}  
 }  
}  
  
*/\*\*  
 \* класс хранящий в ссылочном приватном  
 \* списке информацию о последних 10 (или меньше) сообщениях  
 \*/  
  
class* Story {  
  
 *private* LinkedList<String> story = *new* LinkedList<>();  
  
 */\*\*  
 \* добавить новый элемент в список  
 \** ***@param el*** *\*/  
  
 public void* addStoryEl(String el) {  
 *// если сообщений больше 10, удаляем первое и добавляем новое  
 // иначе просто добавить  
 if* (story.size() >= 10) {  
 story.removeFirst();  
 story.add(el);  
 } *else* {  
 story.add(el);  
 }  
 }  
  
 */\*\*  
 \* отсылаем последовательно каждое сообщение из списка  
 \* в поток вывода данному клиенту (новому подключению)  
 \** ***@param writer*** *\*/  
  
 public void* printStory(BufferedWriter writer) {  
 *if*(story.size() > 0) {  
 *try* {  
 writer.write("History messages" + "\n");  
 *for* (String vr : story) {  
 writer.write(vr + "\n");  
 }  
 writer.write("/...." + "\n");  
 writer.flush();  
 } *catch* (IOException ignored) {}  
  
 }  
  
 }  
}  
  
*public class* Server {  
  
 *public static final int* PORT = 8080;  
 *public static* LinkedList<ServerSomthing> *serverList* = *new* LinkedList<>(); *// список всех нитей - экземпляров  
 // сервера, слушающих каждый своего клиента  
 public static* Story *story*; *// история переписки  
  
 /\*\*  
 \** ***@param args*** *\** ***@throws*** *IOException  
 \*/  
  
 public static void* main(String[] args) *throws* IOException {  
 ServerSocket server = *new* ServerSocket(PORT);  
 *story* = *new* Story();  
 System.out.println("Server Started");  
 *try* {  
 *while* (*true*) {  
 *// Блокируется до возникновения нового соединения:* Socket socket = server.accept();  
 *try* {  
 *serverList*.add(*new* ServerSomthing(socket)); *// добавить новое соединенние в список* } *catch* (IOException e) {  
 *// Если завершится неудачей, закрывается сокет,  
 // в противном случае, нить закроет его:* socket.close();  
 }  
 }  
 } *finally* {  
 server.close();  
 }  
 }  
}

**Client.java**

*import* java.net.\*;  
*import* java.io.\*;  
*import* java.text.SimpleDateFormat;  
*import* java.util.Date;  
  
*/\*\*  
 \* создание клиента со всеми необходимыми утилитами, точка входа в программу в классе Client  
 \*/  
  
class* ClientSomthing {  
  
 *private* Socket socket;  
 *private* BufferedReader in; *// поток чтения из сокета  
 private* BufferedWriter out; *// поток чтения в сокет  
 private* BufferedReader inputUser; *// поток чтения с консоли  
 private* String addr; *// ip адрес клиента  
 private int* port; *// порт соединения  
 private* String nickname; *// имя клиента  
 private* Date time;  
 *private* String dtime;  
 *private* SimpleDateFormat dt1;  
  
 */\*\*  
 \* для создания необходимо принять адрес и номер порта  
 \*  
 \** ***@param addr*** *\** ***@param port*** *\*/  
  
 public* ClientSomthing(String addr, *int* port) {  
 *this*.addr = addr;  
 *this*.port = port;  
 *try* {  
 *this*.socket = *new* Socket(addr, port);  
 } *catch* (IOException e) {  
 System.err.println("Socket failed");  
 }  
 *try* {  
 *// потоки чтения из сокета / записи в сокет, и чтения с консоли* inputUser = *new* BufferedReader(*new* InputStreamReader(System.in));  
 in = *new* BufferedReader(*new* InputStreamReader(socket.getInputStream()));  
 out = *new* BufferedWriter(*new* OutputStreamWriter(socket.getOutputStream()));  
 *this*.pressNickname(); *// перед началом необходимо спросит имя  
 new* ReadMsg().start(); *// нить читающая сообщения из сокета в бесконечном цикле  
 new* WriteMsg().start(); *// нить пишущая сообщения в сокет приходящие с консоли в бесконечном цикле* } *catch* (IOException e) {  
 *// Сокет должен быть закрыт при любой  
 // ошибке, кроме ошибки конструктора сокета:* ClientSomthing.*this*.downService();  
 }  
 *// В противном случае сокет будет закрыт  
 // в методе run() нити.* }  
  
 */\*\*  
 \* просьба ввести имя,  
 \* и отсылка эхо с приветсвием на сервер  
 \*/  
  
 private void* pressNickname() {  
 System.out.print("Press your nick: ");  
 *try* {  
 nickname = inputUser.readLine();  
 out.write("Hello " + nickname + "\n");  
 out.flush();  
 } *catch* (IOException ignored) {  
 }  
  
 }  
  
 */\*\*  
 \* закрытие сокета  
 \*/  
 private void* downService() {  
 *try* {  
 *if* (!socket.isClosed()) {  
 socket.close();  
 in.close();  
 out.close();  
 }  
 } *catch* (IOException ignored) {}  
 }  
  
 *// нить чтения сообщений с сервера  
 private class* ReadMsg *extends* Thread {  
 @Override  
 *public void* run() {  
  
 String str;  
 *try* {  
 *while* (*true*) {  
 str = in.readLine(); *// ждем сообщения с сервера  
 if* (str.equals("stop")) {  
 ClientSomthing.*this*.downService(); *// харакири  
 break*; *// выходим из цикла если пришло "stop"* }  
 System.out.println(str); *// пишем сообщение с сервера на консоль* }  
 } *catch* (IOException e) {  
 ClientSomthing.*this*.downService();  
 }  
 }  
 }  
  
 *// нить отправляющая сообщения приходящие с консоли на сервер  
 public class* WriteMsg *extends* Thread {  
  
 @Override  
 *public void* run() {  
 *while* (*true*) {  
 String userWord;  
 *try* {  
 time = *new* Date(); *// текущая дата* dt1 = *new* SimpleDateFormat("HH:mm:ss"); *// берем только время до секунд* dtime = dt1.format(time); *// время* userWord = inputUser.readLine(); *// сообщения с консоли  
 if* (userWord.equals("stop")) {  
 out.write("stop" + "\n");  
 ClientSomthing.*this*.downService(); *// харакири  
 break*; *// выходим из цикла если пришло "stop"* } *else* {  
 out.write("(" + dtime + ") " + nickname + ": " + userWord + "\n"); *// отправляем на сервер* }  
 out.flush(); *// чистим* } *catch* (IOException e) {  
 ClientSomthing.*this*.downService(); *// в случае исключения тоже харакири* }  
  
 }  
 }  
 }  
}  
  
*public class* Client {  
  
 *public static* String *ipAddr* = "localhost";  
 *public static int port* = 8080;  
  
 */\*\*  
 \* создание клиент-соединения с узананными адресом и номером порта  
 \** ***@param args*** *\*/  
  
 public static void* main(String[] args) {  
 *new* ClientSomthing(*ipAddr*, *port*);  
 }  
}

**Вывод программы**

****

****

**Вывод**

В результате выполнения ознакомились с базовыми технологиями реализации конечной точки для передачи и приема данных по сети - сокетом