|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий

Кафедра Инструментального и прикладного программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №3**

по дисциплине «Архитектура клиент-серверных приложений»

**Тема практической работы: Программирование JAVA сокетов**

**Студент группы** ИКБО-10-19 Харитонов Дмитрий Михайлович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись студента)

**Руководитель практической работы** преподаватель Степанов П. В.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись руководителя)

Работа представлена «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Допущен к работе «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Москва 2021

**Цель:**

Знакомство студентов с одной из базовых технологии реализации конечной точки для передачи и приема данных по сети - сокетом.

**Задание:**

Необходимо создать клиент-серверное приложение на языке JAVA с использованием socket, для широковещательного общения пользователей. Приложение может быть как консольным, так и оснащённым полноценным GUI. Клиентское приложение считывает данные из стандартного ввода и отсылает сообщение серверу (с помощью TCP/IP). Сервер, в свою очередь, накапливает сообщения и раз в 5 секунд осуществляет массовую рассылку всем клиентам. Если сообщений за указанный период не поступило, то рассылка не производится. Клиент, получивший сообщение, отображает на экране текст данного сообщения. Структуру и поведение данного клиент-серверного приложения, в том числе, **например**, в части регистрации конкретного клиента и формата широковещательного сообщения, студент определяет самостоятельно.

**Ход работы:**

В ходе выполнения практической работы были созданы необходимые классы. Ниже, на рисунках, представлена реализация программы и результат её работы.

Исходный код программы:

Client.java

import java.net.\*;  
import java.io.\*;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
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;  
  
  
 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 {  
 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) {  
 ClientSomthing.this.downService();  
 }  
 }  
  
 private void pressNickname() {  
 System.*out*.print("Press your nick: ");  
 try {  
 nickname = inputUser.readLine();  
 out.write("Hello " + nickname + "\n");  
 out.flush();  
 } catch (IOException ignored) {  
 }  
  
 }  
  
 private void downService() {  
 try {  
 if (!socket.isClosed()) {  
 socket.close();  
 in.close();  
 out.close();  
 }  
 } catch (IOException ignored) {}  
 }  
  
 private class ReadMsg extends Thread {  
 @Override  
 public void run() {  
  
 String str;  
 try {  
 while (true) {  
 str = in.readLine();   
 if (str.equals("stop")) {  
 ClientSomthing.this.downService();  
 break;   
 }  
 System.*out*.println(str);  
 }  
 } catch (IOException e) {  
 ClientSomthing.this.downService();  
 }  
 }  
 }  
  
   
 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");   
 dtime = dt1.format(time);  
 userWord = inputUser.readLine();   
 if (userWord.equals("stop")) {  
 out.write("stop" + "\n");  
 ClientSomthing.this.downService();   
 break;   
 } else {  
 out.write("(" + dtime + ") " + nickname + ": " + userWord + "\n");   
 }  
 out.flush();  
 } catch (IOException e) {  
 ClientSomthing.this.downService();   
 }  
  
 }  
 }  
 }  
}  
  
public class Client {  
  
 public static String *ipAddr* = "localhost";  
 public static int *port* = 8080;  
  
 public static void main(String[] args) {  
 new ClientSomthing(*ipAddr*, *port*);  
 }  
}  
class Client2 {  
  
 public static String *ipAddr* = "localhost";  
 public static int *port* = 8080;  
 public static void main(String[] args) {  
 new ClientSomthing(*ipAddr*, *port*);  
 }  
}

Server.java

import java.io.\*;  
import java.net.\*;  
import java.util.LinkedList;  
  
  
class ServerSomthing extends Thread {  
  
 private Socket socket;  
 private BufferedReader in;  
  
 public BufferedWriter getOut() {  
 return out;  
 }  
  
 private BufferedWriter out;  
  
 public ServerSomthing(Socket socket) throws IOException {  
 this.socket = socket;  
 in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
 out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));  
 Server.*story*.printStory(out);  
 start();  
 }  
 @Override  
 public void run() {  
 String word;  
 try {  
 word = in.readLine();  
 try {  
 out.write(word + "\n");  
 out.flush();  
 } catch (IOException ignored) {}  
 try {  
 while (true) {  
 word = in.readLine();  
 if(word.equals("stop")) {  
 this.downService();  
 break;  
 }  
 System.*out*.println("Echoing: " + word);  
 Server.*story*.addStoryEl(word);  
 }  
 } catch (NullPointerException ignored) {}  
  
  
 } catch (IOException e) {  
 this.downService();  
 }  
 }  
  
 private void send(String msg) {  
 try {  
 out.write(msg + "\n");  
 out.flush();  
 } catch (IOException ignored) {}  
  
 }  
  
 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) {}  
 }  
}  
  
  
class Story {  
  
 public LinkedList<String> getStory() {  
 return story;  
 }  
  
 private LinkedList<String> story = new LinkedList<>();  
  
  
 public void addStoryEl(String el) {  
 if (story.size() >= 100) {  
 story.removeFirst();  
 story.add(el);  
 } else {  
 story.add(el);  
 }  
 }  
  
 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*;   
   
 public static void main(String[] args) throws IOException, InterruptedException {  
 ServerSocket server = new ServerSocket(*PORT*);  
 *story* = new Story();  
 System.*out*.println("Server Started");  
 try {  
 for (int i=0;i<2;i++){  
 // Блокируется до возникновения нового соединения:  
 Socket socket = server.accept();  
 try {  
 *serverList*.add(new ServerSomthing(socket)); // добавить новое соединенние в список  
 } catch (IOException e) {  
 socket.close();  
 }  
  
 }  
 while(true){  
 for (ServerSomthing vr : Server.*serverList*) {  
 BufferedWriter out=vr.getOut();  
 *story*.printStory(out);  
 }  
 *story*.getStory().clear();  
 Thread.*sleep*(5000);  
 }  
 } finally {  
 server.close();  
 }  
 }  
}

**Вывод программы:**

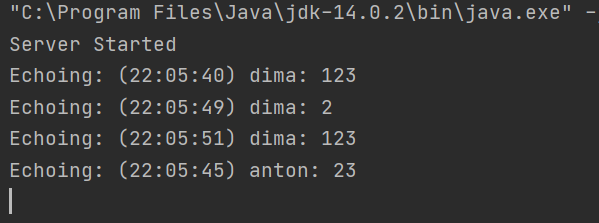


Рисунок 1 – результат выполнения программы

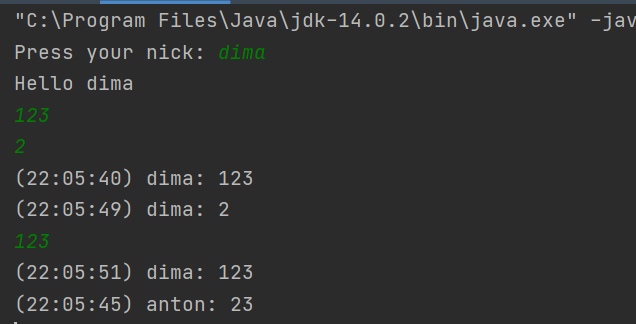


Рисунок 2 – результат выполнения программы

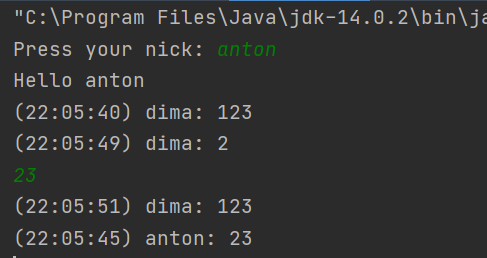


Рисунок 3 – результат выполнения программы

**Вывод:**

В результате выполнения практической работы были получены знания и навыки работы c socket в JAVA.

**GitHub:**

Github - https://github.com/Unripper1/Architecture