|  |
| --- |
| https://lh6.googleusercontent.com/QcftzNtI05T0Y6fjdSh1Rr2rt8oqZ1IvnLvbn1jLJ7CCyteVir3k-xBLv4SL1wAgWJsRhmmJSR0UW-RP63_GQenE4vVWv05BRoZTsmIcBccVTnfxwmsnNMvjg599x9SqZd8E3dkd |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждениевысшего образования«МИРЭА - Российский технологический университет»РТУ МИРЭА |

Институт Информационных Технологий (ИТ)

Кафедра инструментального и прикладного программного обеспечения (ИППО)

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №6** | |
| **по дисциплине** | |
| «Разработка клиент-серверных приложений» | |
|  | |
| Выполнил студент группы ИКБО-01-18 | Маркин В.В. |
| Принял ассистент кафедры ИППО | Строганкова Н.В. |

|  |  |  |
| --- | --- | --- |
| Практическая работа выполнена | «\_\_»\_\_\_\_\_\_\_2020 г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_2020 г. | *(подпись руководителя)* |

**Содержание**

[1. Цель практической работы 3](#_Toc528748841)

[2. Задание 3](#_Toc528748842)

[3. Поэтапное выполнение практической работы 3](#_Toc528748843)

[5. Текст разметки 4](#_Toc528748844)

[6. Результат работы 9](#_Toc528748845)

[7. Вывод 10](#_Toc528748846)

[Список использованных источников 11](#_Toc528748847)

# 1 Цель практической работы

Целью данной практической работы является ознакомление со средой разработки IntellijIDEA и написания простейшего серверного приложения на языке программирования Java, которое потом может быть использовано в качестве основы для дальнейших практических работ.

# 2 Задание

Найти максимальное значение среди множества чисел (без использования класса Math). Числа должны поступать в виде строки с некоторым разделителем (пример: «11, 32, 1, 22, 14»); в массиве; списком чисел.

# 3 Поэтапное выполнение практической работы

1) Используя класс java.net.ServerSocket, написать простейший echo-сервер. Технические требования, предъявляемые к серверу:

* а) порт, на котором запускается сервер — 12345. Транспортный протокол — TCP;
* б) количество одновременно подключенных клиентов – 1. То есть использование потоков (экземпляров класса Thread) на первом этапе не предусматривается;
* в) сервер должен обеспечивать echo-функционал, то есть просто передавать обратно клиенту полученный буквенный символ;
* г) на консоль (с помощью метода System.out.println()) должны выводиться основные этапы работы программы (клиент установил соединение, был получен и передан обратно символ, соединение разорвано). 10

2) Изменить написанный класс echo-сервера. Сервер должен производить вычисления в соответствии с индивидуальным заданием, выполненным в пункте 4.4. В ответ клиент должен получить результат выполненной операции.

3) Написать клиентскую часть Socket:

* а) клиент должен будет подключиться к созданному ранее серверу;
* б) клиенту нужно передать данные параметров в соответствии с индивидуальным заданием, в ответ получить результат действий.

4) Произвести сборку серверной и клиентской части в jar-файл.

5) Запустить jar-файлы серверной и клиентской части.

# 4 Текст разметки

Серверная часть

|  |
| --- |
| Listener.java |
| package server;  import java.io.DataInputStream;  import java.io.DataOutputStream;  import java.io.IOException;  import java.net.ServerSocket;  import java.net.Socket;  public class Listener {  int port;  DataOutputStream out;  public Listener(int port) {  this.port = port;  }  public void start() throws IOException, InterruptedException {  String text = "";  System.out.print("Сервер стартанул. Ждем соединения...");  try (ServerSocket server = new ServerSocket(port)) {  // становимся в ожидание подключения к сокету под именем - "client" на серверной стороне  Socket client = server.accept();  // после хэндшейкинга сервер ассоциирует подключающегося клиента с этим сокетом-соединением  System.out.print("Соединение установлено");  // инициируем каналы для общения в сокете, для сервера  // канал записи в сокет  out = new DataOutputStream(client.getOutputStream());  System.out.println("канал записи создан");  // канал чтения из сокета  DataInputStream in = new DataInputStream(client.getInputStream());  System.out.println("канал чтения создан");  // начинаем диалог с подключенным клиентом в цикле, пока сокет не закрыт  while (!client.isClosed()) {  System.out.println("Сервер готов читать из сокета");  // сервер ждёт в канале чтения (input stream) получения данных клиента  String entry = in.readUTF();  // после получения данных считывает их  System.out.println("Клиент прислал: " + entry);  // если клиент прислал кодовое слово quit - сервер выходит из цикла и закрывает все сокеты и соединения  if (entry.equalsIgnoreCase("quit")) {  //отвечаем эхом  sendEcho(entry);  // засыпаем на 3 сек  Thread.sleep(3000);  break;  }  text = text.concat(entry);  Math math = new Math(text);  math.Parser();  System.out.println("Максимальное число в последовательности: "+ math.getResult());  // System.out.println("Набираем:"+text);  text="";  sendEcho(entry);  }  // если условие выхода - верно выключаем соединения  System.out.println("Клиент отключился");  System.out.println("Закрываем соединения и сокеты");  // закрываем сначала каналы сокета !  in.close();  out.close();  // потом закрываем сам сокет общения на стороне сервера!  client.close();  // потом закрываем сокет сервера который создаёт сокеты общения  // хотя при многопоточном применении его закрывать не нужно  // для возможности поставить этот серверный сокет обратно в ожидание нового подключения  System.out.println("Пока!");  } catch (IOException e) {  e.printStackTrace();  }  }  private void sendEcho(String text) throws IOException {  System.out.println("Отправлям эхо клиенту: "+text);  out.writeUTF(text);  // освобождаем буфер сетевых сообщений (по умолчанию сообщение не сразу отправляется в сеть, а сначала накапливается в специальном буфере сообщений, размер которого определяется конкретными настройками в системе, а метод - flush() отправляет сообщение не дожидаясь наполнения буфера согласно настройкам системы  out.flush();  }  } |

|  |
| --- |
| Math.java |
| package server;  import java.util.ArrayList;  import java.util.List;  public class Math implements Result{  Double max;  String text;  double[] nums;  List<Double> lineOfNums;  private int i;  /\*  Создание метода для возвращения результата.  Входных параметров: нет  Выходные данные: результат вычислений.  \*/  @Override  public double getResult() {  //Вернуть параметр - результат вычислений.  return max;  }  //Конструктор для строки  public Math(String text) {  this.text = text;  System.out.println("Конструктор для строки:"+this.text);  }  public void Parser(){  ArrayList<Double>list = new ArrayList<>();  //Массив чисел  String[] temp = text.split(",");  int length = temp.length;  //Парсим числа в целочисленные и добавление в лист  for(int j = 0; j < temp.length; j++){  // i = Integer.parseInt(temp[j]);  double k = Double.parseDouble(temp[j]);  list.add(k);  }  //Отправляем в метод сортировки  //Без ожидания  maxNumOfList(list);  }  public void maxNumOfList(ArrayList<Double>list){  max = list.get(0);  for (int i = 1; i < list.size(); i++){  if(list.get(i) > max){  max = list.get(i);  }  }  }  } |

|  |
| --- |
| Result.java |
| package server;  public interface Result {  double getResult();  } |

|  |
| --- |
| ServerStart.java |
| package server;  import java.io.IOException;  public class ServerStart {  public static void main(String[] args){  Listener listener = new Listener(12345);  try {  listener.start();  } catch (Exception e) {  e.printStackTrace();  }  }  } |

Клиентская часть

|  |
| --- |
| ClientStart.java |
| package client;  public class ClienStart {  public static void main(String[] args) {  ConsoleSender consoleSender = new ConsoleSender("localhost", 12345);  try {  consoleSender.start();  } catch (Exception e) {  e.printStackTrace();  }  }  } |

|  |
| --- |
| ConsoleSender.java |
| package client;  import java.io.\*;  import java.net.Socket;  public class ConsoleSender {  String host;  int port;  public ConsoleSender(String host, int port) {  this.host = host;  this.port = port;  }  public void start() throws IOException, InterruptedException {  System.out.println("Клиент стартанул");  // запускаем подключение сокета по известным координатам и нициализируем приём сообщений с консоли клиента  try (Socket socket = new Socket(host, port)) {  // консоль  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  // сокет для записи данных на сервер  DataOutputStream oos = new DataOutputStream(socket.getOutputStream());  // сокет для чтения ответа с сервера  DataInputStream ois = new DataInputStream(socket.getInputStream());  System.out.println("Клиент подключился к сокету");  System.out.println("Введите данные...");  // проверяем живой ли канал с сервером  while (!socket.isOutputShutdown()) {  // ждём консоли клиента на предмет появления в ней данных  if (br.ready()) {  // данные появились - работаем  String clientCommand = br.readLine();  // пишем данные с консоли в канал сокета для сервера  oos.writeUTF(clientCommand);  oos.flush();  System.out.println("Отправили на сервер: " + clientCommand);  // проверяем условие выхода из соединения  if (clientCommand.equalsIgnoreCase("quit")) {  // если условие выхода достигнуто разъединяемся  System.out.println("Пришла команда завершения");  // смотрим что нам ответил сервер на последок перед закрытием ресурсов  String in = ois.readUTF();  System.out.println(in);  // после предварительных приготовлений выходим из цикла записи чтения  break;  }  // если условие разъединения не достигнуто продолжаем работу  System.out.println("Данные отправлены - ждем ответа с сервера...");  // если успел забираем ответ из канала сервера в сокете и сохраняем её в ois переменную, печатаем на свою клиентскую консоль  System.out.print("читаем... ");  String in = ois.readUTF();  System.out.println(in);  System.out.println("Введите данные...");  }  }  // на выходе из цикла общения закрываем свои ресурсы  System.out.println("Закрываем соединения и сокеты");  }  }  } |

# 5 Результат работы

В процессе выполнения работы были разработаны Клиентская и Серверная части. Также была добавлена возможность выхода из приложения с помощью ввода ключевого слова «quit» в клиентский терминал. Серверный терминал только печатает данные.

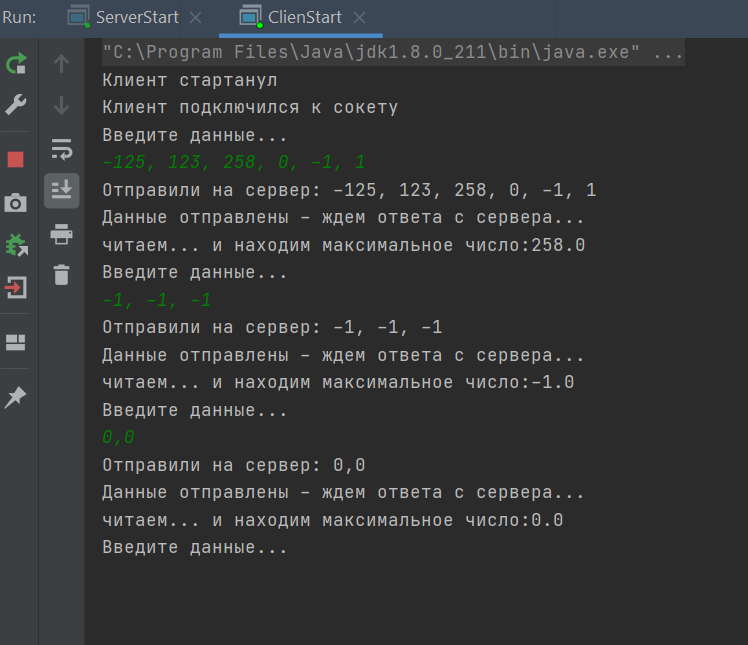


Рис. 1. Клиентская часть приложения

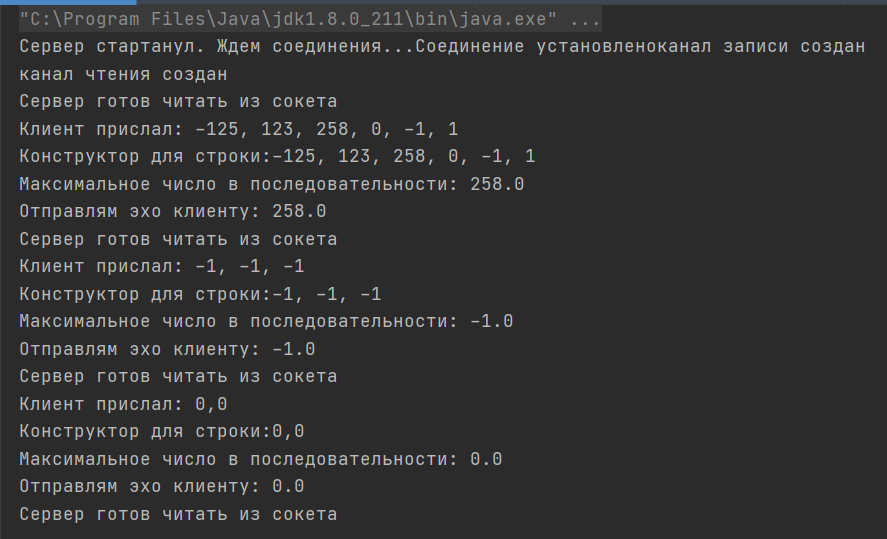


Рис. 2. Серверная часть приложения

# 6 Вывод

В процессе выполнения данной работы, мы получили практические навыки работы со средой разработки IntellijIDEA. Также научились создавать простейшее серверное приложения на языке программирования Java, которое потом может быть использовано в качестве основы для дальнейших практических работ.

# Список использованных источников

1. <https://habr.com/ru/post/330676/>
2. <http://mindhalls.ru/client-server-application-in-java/>
3. <https://hr-vector.com/java/klient-server>
4. <https://javarush.ru/groups/posts/2529-chastjh-5-servletih-pishem-prostoe-veb-prilozhenie>