Титульный лист материалов по дисциплине

ДИСЦИПЛИНА **Разработка клиент-серверных приложений**

полное название дисциплины без аббревиатуры

ИНСТИТУТ **Информационных технологий**

КАФЕДРА **Инструментального и прикладного программного**

полное название кафедры **обеспечения**

ГРУППА/Ы **ИКБО-01/02/03/12/13/16-18**

номер групп/ы, для которых предназначены материалы

ВИД УЧЕБНОГО МАТЕРИАЛА **Материал к практическим занятиям**

лекция; материал к практическим занятиям; контрольно-измерительные материалы к практическим занятиям; руководство к КР/КП, практикам

ПРЕПОДАВАТЕЛЬ **Строганкова Наталья Владимировна**

фамилия, имя, отчество

СЕМЕСТР **5**

указать номер семестра обучения

ПРАКТИЧЕСКАЯ РАБОТА № 1

1 Цель 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 Теоретическая часть

Любое клиент-серверное приложение работает через сокеты. Для создания интернет-приложения на Java обычно используются два класса: java.net.Socket и java.net.ServerSocket, с помощью которых и осуществляется передача данных по сети. В языке Java (точно так же, как во многих других языках программирования высокого уровня) понятия «серверная сокета» и «клиентская сокета» разделены и обеспечиваются разными классами.

3 Описание класса Socket

Этот класс реализует клиентские сокеты. Сокета является конечной точкой для передачи данных между двумя машинами.

Основные конструкторы класса Socket:

– **Socket (InetAddressaddress, intport)** — создает экземпляр класса Socket с входящими параметрами IP-адрес и порт той машины, с которой мы связываемся;

– **Socket (Stringhost, intport, InetAddresslocalAddr, intlocalPort)** — более точная настройка сокета — создает экземпляр класса Socket с входящими данными об IP-адресе и порте локальной и удаленной машины;

– **Socket (InetAddresshost, intport, booleanstream)** — позволяет настроить сокету на передачу данных по протоколу TCP/IP (stream = true) или UDP (stream = false), создает экземпляр класса Socket с входящими параметрами IP-адрес и порт той машины, с которой мы связываемся, и параметром stream.

3.1 Основные методы класса Socket

Основные методы класса Socket:

– **getInputStream()** — метод, который возвращает экземпляр класса InputStream, отвечает за входящий поток данных сокеты;

– **getOutputStream()** — метод, который возвращает экземпляр класса OutputStream, отвечает за исходящий поток данных сокеты;

– **close()** — метод завершения работы сокеты, после выполнения этого метода дальнейшее взаимодействие с объектом Socket невозможно.

3.2 Описание класса ServerSocket

Этот класс реализует сокеты сервера. Сокета сервера ожидает запросы от клиентского приложения, который обращается к ней либо через сеть, либо с того же компьютера. Сокета выполняет некоторую работу, основанную на том запросе, и затем возвращает результат запрашивающей стороне.

Основные конструкторы класса ServerSocket:

– **ServerSocket (intport)** — создает экземпляр класса ServerSocket, с входным параметром порт (локальной машины), при этом порт должен быть свободен, иначе произойдет событие ошибки;

– **ServerSocket (intport, intbacklog, InetAddressbindAddr)** — создает экземпляр класса ServerSocket, с входными параметрами порт, IP-адрес (bindAddr) и количеством возможных подключений (backlog), при этом порт должен быть свободен, иначе произойдет событие ошибки.

3.3 Основные методы класса ServerSocket

Основные методы класса ServerSocket:

– **accept()** — возвращает экземпляр класса Socket, с которым в дальнейшем будет работа по обмену данными. Данный метод ожидает подключения к созданной ServerSocket (к открытому порту, описанному при создании ServerSocket), пока подключение не произошло, находится в состоянии ожидания;

– **close()** — метод завершения работы серверной сокеты, после выполнения этого метода дальнейшее взаимодействие с объектом ServerSocket невозможно.

Обычно программная часть сервера и клиента не сильно отличаются друг от друга:

Сервер:

```
ServerSocketserverSocket = new ServerSocket(12345);
Socket socket = serverSocket.accept();
```

Клиент:

```
Socketsocket = newSocket ("localhost",12345);
```

Дальнейшее взаимодействие осуществляется непосредственно с объектом сокеты.

3.4 Работа с входящим и исходящим потоком байт

Основные методы, необходимые для сокеты — это передача данных, то есть получение и отправление потока данных:

– для получения потока входящих данных нужно использовать метод getInputStream(). InputStream — абстрактный класс, задающий используемую в Java модель входных потоков;

– для отправки исходящего потока данных нужно использовать метод getOutputStream(). OutputStream — абстрактный класс. Он задает модель выходных потоков Java.

getInputStream() и getOutputStream() возвращают поток байт, с которыми не очень удобно, для этого существует множество стандартных классов, которые помогают представить входящий (исходящий) поток в более удобном для нас формате, например: в виде String, class,..

Перечислим основные классы, которые могут использоваться для преобразования потока байт в другой формат и выступать в качестве обертки: ObjectInputStream и ObjectOutputStream позволяют передавать/получать ранее сериализированные объекты, то есть мы можем передавать ранее сформированный экземпляр класса и получить его на другой стороне как структурированный объект.

Пример:

Листинг класса Student (хранит два значения: имя, возраст; и метод getTitle()) возвращает некоторую структурированную информацию о содержании класса. Данный класс реализует интерфейс Serializable.Serialization, предоставляет стандартный механизм для создания сериализуемых объектов.

Сериализация — это процесс сохранения состояния объекта в последовательность байт.

Пример POJO-класса Student:

```
1.  Public class Student implements Serializable {
2.  Private String name;//Имя
3.  private Integer age;//Возраст
4.  public String getName() {
5.  return name;
6.  }
7.  public void setName(String name) {
8.  this.name = name;
9.  }
10. public Integer getAge() {
11. return age;
12. }
13. public void setAge (Integer age) {
14. this.age = age;
15. }
16. public String getTitle() {
17. return getName()+" "+getAge();
18. }
19. }
```

Действия отправителя объекта через сокету.

При отправке объекта изначально его необходимо создать и заполнить данными. Для этого создается экземпляр класса Student и заполняется данными.

```
1. Student student = new Student();
2. student.setAge (18);
3. student.setName("Mr.Smith");
```

После того как экземпляр класса будет создан, его можно будет отправлять, для чего используем класс ObjectOutputStream. Этот класс позволяет отправлять данные объектом, который реализует интерфейс java.io.Serializable. Создание переменой, присвоенной экземпляру класса ObjectOutputStream, с помощью которой в дальнейшем будет осуществляться передача данных. ObjectOutputStream использует в качестве ресурса выходной поток данных ранее созданной сокеты.

```
1. ObjectOutputStream out = new ObjectOutputStream (socket.
getOutputStream());
2. out.writeObject (student);//происходит отправление объекта Действия
принятия объекта через сокету.
```

Для того чтобы принять объект, нужно использовать класс ObjectInputStream, – он позволяет входящие данные принимать в виде объекта, а не потока байт. Для этого создаем экземпляр класса ObjectInputStream, а в качестве входного параметра используем входной поток данных ранее созданной сокеты:

```
1.  ObjectInputStream in = new ObjectInputStream
(socket.getInputStream());
2.  try {
3.  Student student = (Student)in.readObject();//получение объекта
Student
4.  System.out.println (student.getTitle());//использование метода
getTitle() класса Student, для вывода результатов на консоль
5.  } catch (ClassNotFoundException e) {
6.  e.printStackTrace();
7.  }
```

В результате выполнения данных операций на экране принимающей стороны должна появиться надпись “Mr.Smith 18”.

3.5 Работа с входным потоком данных как со строками

InputStreamReader является мостом между потоком байт и потоком символов. Несет в себе информацию о кодировке (сам производит кодировку и декодировку, например, UTF-8). Для создания экземпляра класса InputStreamReader на вход необходимо подать класс InputStream, например, System.in или socket.getInputStream.

BufferedReader буферизует символы и позволяет извлекать из потока как сформированные строки, так и просто символы. Для создания экземпляра класса необходимо поместить в него любой класс, реализующий абстрактный класс Reader. Например, InputStreamReader, StringReader.

Пример:

```
1.  InputStreamReader isr = new InputStreamReader (System.in);
//преобразование входного потока байт
2.  BufferedReader br = new BufferedReader (isr);//буферизация символов
3.  while (true){
4.  String st = br.readLine();//чтение из буфера строку
5.  if (st==null)
6.  {
7.  break;
8.  }
9.  }
```

3.6 Работа с исходящим потоком данных как со строками

PrintWriter позволяет введенный печатный текст представить к байтовому потоку вывода. Для создания экземпляра класса необходимо иметь входные параметры наследников абстрактного класса Writer (например, OutputStreamWriter) или классов, реализующих интерфейсы Closeable, Flushable (например, OutputStream).

Пример:

```
1.  PrintWriter printWriter = new PrintWriter (socket.getOutput
Stream(), true);
2.  printWriter.println("Введите значение a:";
```

4 Порядок выполнения работы

4.1 Подготовка рабочего места

1) Для выполнения практических заданий необходимо установить на персональном компьютере следующие средства разработки:

а) Java Development Kit (JDK);

б) IntelliJIDEA (CommunityEdition, свободно распространяемый вариант);

в) СУБД на выбор: MS SQL Server, MySQL, PostgreSQL, Oracle.

2) Создать новый пустой проект (File –> NewProject…). Записать его в каталог, название которого соответствует вашей фамилии, номеру группы и варианту задания (например, «Иванов_ИКБО0118_вар4»):

а) при создании проекта необходимо указать, какое JDK вы будете использовать; если его не будет в предложенном меню, то вручную укажите путь до него (например, C:\Program Files\Java\jdk1.7.0_15).

3) Создать пустой класс, содержащий статический метод выполнения (JavaApplication). Реализация метода может быть примерно следующей:

```
public static void main (String argc []) {
System.out.println (“Hello, World!”);
}
```

4) Запустить созданную программу на выполнение, убедиться, что на консоль (стандартный поток вывода) выводится надпись.

4.2 Требования к выполнению индивидуального задания

Требования к выполнению индивидуального задания следующие:

– индивидуальное задание должно быть оформлено в отдельном классе;

– класс должен реализовывать созданный вами интерфейс Result, который будет иметь один метод, возвращающий строку результата вычислений getResult () без входных параметров;

– все входные параметры должны поступать в класс с помощью перегрузок конструкторов и должны совпадать с индивидуальным заданием (пример: в задании 1 должно быть описано три перегруженных конструктора, которые будут принимать String, Double [], List<Double>);

– в классе должно быть описано поле, в котором будут храниться распарсенные значения (пример: в задание 1 в конструктор класса поступает обычная строка «11, 32, 1, 22, 14», конструктор должен разбить данную строку по запятой на массив чисел и заполнить поле, с которым в дальнейшем мы будем работать для вычисления максимального значения);

– для отладки и проверки работоспособности выполненного индивидуального задания создаем отдельный класс с методом main(), который будет взаимодействовать с созданным ранее классом (с классом, в котором выполнено индивидуальное задание) и позволит вводить данные (в консоли) для вычисления, после чего вывести результат на экран.

4.3 Порядок выполнения индивидуального задания

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).

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.

4.4 Варианты индивидуальных заданий

Индивидуальные задания выбираются в соответствии с порядковым номером студента в алфавитном списке группы. Если заданий меньше, чем студентов, то список заданий «зацикливается», то есть студент под номером 16 берет задание 1, 17 – 2, 18 – 3 и т.д.

Варианты индивидуальных заданий:

1) 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.

2) Сортировка списка слов, без использования сторонних классов сортировки, например, Collections метод sort(). Слова должны поступать сплошным текстом с разделителем; списком; отдельными значениями данных.

3) Подсчет одинаковых слов. Слова должны поступать сплошным текстом и с разделителем; списком; отдельными значениями данных.

4) Подсчет четных и нечетных символов. Числа должны поступать в виде строки с некоторым разделителем (пример: «11, 32, 1, 22, 14»); в массиве; списком чисел.

5) Конвертер систем счисления. На вход поступает число, которое мы хотим конвертировать; система счисления конвертируемого числа; система счисления, в

которую мы хотим преобразовать число. Числа должны поступать в виде строки с некоторым разделителем; в массиве; списком чисел.

6) Вывести уравнение прямой, проходящей через две точки. На вход поступает 4 числа: x1, y1, x2, y2. Числа должны поступать в виде строки с некоторым разделителем; в массиве; списком чисел.

7) Дан текст. Определите процентное отношение строчных и прописных букв к общему числу символов в нем. На вход поступает текст в виде строки.

8) Дана строка, состоящая из слов и чисел, отделенных друг от друга разделяющим символом (при этом один символ будет служебным, например, «.», который будет характеризовать вещественные числа). Сформировать три строки, одна из которых содержит только целые числа, встречающиеся в исходной строке, вторая — только вещественные числа, а третья — оставшиеся слова. Текст должен поступать сплошным текстом с разделителем.

9) Подсчет одинаковых символов. Символы должны поступать сплошным текстом с разделителем; списком; отдельными значениями данных.

10) Наибольший общий делитель (НОД) чисел (пример: 3430 и 1365 — это 35. Другими словами, 35 — наибольшее число, на которое и 3430 и 1365 делятся без остатка). Числа должны поступать в виде строки с некоторым разделителем; в массиве; списком чисел.

11) Для каждого натурального числа в промежутке от m до n вывести все делители. Числа должны поступать в виде строки с некоторым разделителем; в массиве; отдельными значениями данных.

12) Перевод римских чисел в арабские (например, XIIV — 13). Поступает в виде строки.

13) Калькулятор. Формула должна поступать в виде в виде текста (пример: «4/2»); отдельными значениями данных.

14) Перемножение матриц. Числа должны поступать в виде строки с некоторым разделителем; в массиве; отдельными значениями данных.

15) Пользователь вводит несколько целых чисел, представляющих собой последовательность. Требуется ее оценить: является ли последовательность

возрастающей; есть ли в ней одинаковые элементы; является ли она знакочередующейся (положительные и отрицательные числа чередуются). Числа должны поступать в виде строки с некоторым разделителем; в массиве; отдельными значениями данных.