

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ТУЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

Институт *прикладной математики и компьютерных наук*

Кафедра *«Информационная безопасность»*

**МЕТОДИЧЕСКИЕ УКАЗАНИЯ
ПО КОНТРОЛЬНО-РАСЧЕТНОМУ ЗАДАНИЮ**

по дисциплине

«СЕТЕВОЕ ПРОГРАММИРОВАНИЕ»

Уровень профессионального образования: *высшее образование –
бакалавриат*

Направление: *09.03.03 Прикладная информатика*

Профиль: *Прикладная информатика в экономике*

Квалификация выпускника: *бакалавр*

Форма обучения: *заочная*

Тула, 2018

ВВЕДЕНИЕ

Java – объектно-ориентированная платформно-независимая многопоточная среда программирования. *Клиент-серверные возможности языка Java.* Язык JAVA разработан для распределенной среды, предоставляет специальные классы и широкие возможности организации работы в сети. Он поддерживает протоколы TCP/IP и, фактически, снижает сложность доступа к сетевому ресурсу до уровня сложности доступа к файлу, а также обеспечивает ряд технологий для разработки сетевых приложений. Кроме того, JAVA позволяет создавать интерактивные сетевые программы за счет поддержки многопоточного программирования и эффективных решений распараллеливания и синхронизации процессов. JAVA обеспечивает разработчика различными инструментальными средствами защиты, для разработки средств обеспечения политики защиты информации, шифрования данных и т.п.

Java - интерпретируемый язык. Сначала программист компилирует исходный текст утилитой `javac` из набора инструментов JDK в Java в байт-коды. Байт-коды являются двоичными и не зависят от архитектуры компьютера (или от платформы). Байт-коды - не законченное приложение, они не выполняются в операционной среде выполнения программ (Javaruntimeenvironment). Обычно в роли Среды выступает браузер или JVM (виртуальная машина Java). Поскольку каждая Среда выполнения создается для конкретной платформы, законченный программный продукт будет работать на этой платформе.

Java - объектно-ориентированный язык. Язык Java является объектно-ориентированным и, следовательно, относится к группе языков, рассматривающих данные как объекты и методы, используемые для этих объектов. Java язык со строгой типизацией, что помогает выявлять возможные скрытые ошибки. В Java отсутствует множественное наследование и указатели, что с одной стороны повышает, читаемость, надежность программного кода, упрощает программирование и предохраняет от множества трудно диагностируемых ошибок, а с другой, несколько ограничивает в ряде ключевых возможностей.

Java обеспечивает очень развитую объектно-ориентированную технологию поддержки обработки особых «исключительных» ситуаций. В Java можно создавать совершенно «абстрактные классы», называемые интерфейсами(*interface*). Интерфейсы позволяют описывать методы, разделяемые и реализуемые между несколькими классами, не учитывая при этом то, как другие классы используют данные методы.

Виртуальная Java-машина. Основным языком Java является виртуальная Java-машина (JavaVirtualMachine, JVM). JVM - это виртуальный компьютер (модель компьютера), располагающийся только в оперативной памяти. JVM позволяет выполнять Java-приложения на множестве платформ, а не только в той системе, для которой скомпилирован код. Возможность компиляции Java-программ для JVM обеспечивает уникальность языка. Но для того чтобы приложения Java выполнялись на конкретной платформе, необходимо реализовать JVM для данной платформы (для каждой платформы своя JVM, что и обеспечивает мобильность JAVA).

Установка и основные утилиты Java. Программа установки Java (пакет Java SDK, известный также под названием JDK) представляет собой самораспаковывающийся архив, при разархивировании которого выдаются инструкции по установке. Не следует разархивировать файлы classes.zip, они предназначены для использования в среде выполнения Java. Java SDK обычно включает 6 подкаталогов:

- bin – содержит выполняемые модули и утилиты JDK;
- demo – включает множество апплетов, а также примеры текстов программ на java;
- docs – содержит документацию по java;
- include – включает заголовочные файлы C и C++, используемые для построения среды Java;
- lib – библиотеки и архивы, используемые в Java;
- src – исходные коды библиотек, созданных компанией Sun.

Наиболее важными в JAVA являются следующие утилиты:

- *javac* – компилятор java, компилирует разработанный файл исходного текста программы в файл (байткод) с таким же именем и расширением .class;
- *java* – утилита-интерпретатор (вернее программа запуска приложений java);
- *appletviewer* – программа просмотра апплетов;
- *jdb* – утилита тестирования программ, написанных на java;
- *javadoc* – утилита для создания документации. Собирает в виде HTML-страницы комментарии из исходного текста программы java, ограниченные комментирующими скобками `/** ... */`.

ЧАСТЬ 1

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Цель: Создание клиент-серверных приложений с использованием протокола TCP/IP

Основные задачи, которые реализуются при организации приложений архитектуры клиент-сервер:

1) Получить информацию с одной машины и переместить её на другую машину. Выполняется с помощью простейшего сетевого программирования.

2) Подключиться к базе данных, находящейся в сети. Выполняется с помощью библиотеки `JavaDataBaseConnectivity (JDBC)`.

3) Представление услуг через web-сервер. Достигается через какой-либо web-сервер, который поддерживает технологию Java-сервлетов и `JavaServerPages (JSP)`.

4) Удалённый запуск методов объектов Java на других компьютерах. Выполняется с помощью `RemoteMethodInvocation (RMI)` – удалённый вызов методов.

5) Использовать код, написанный на других языках и работающий в других архитектурах. Позволяет технология `CORBA (CommonObjectRequestBrokerArchitecture)` – общая архитектура брокеров объектных запросов.

6) Изолировать бизнес-логику от соединения, особенно соединения с БД, включающего обработку транзакций и безопасность в корпоративных клиент-серверных системах. Реализуется при помощи `EJB (EnterpriseJavaBeans)`.

7) Простое динамическое добавление, использование и удаление устройств бытового оборудования и программного обслуживания, обеспечивающих определенный сервис для клиента сети. Обеспечивается при помощи `Ini`.

Сервера и клиенты

Различие между сервером и клиентом существенно не только тогда, когда клиент пытается подключиться к серверу. Как только они соединятся, происходит процесс двухстороннего общения и не важно, что один является сервером, а другой - клиентом.

Итак, работа сервера - слушать соединение, и это выполняется с помощью специального создаваемого серверного объекта (сокета), который равен: `сокет=IP+номер_порта`. Работа клиента - попытаться создать соединение с сервером, что выполняется с помощью специального клиентского объекта (сокета). Как только соединение установлено,

соединение превращается в потоковый объект ввода/вывода, и с этого момента можно рассматривать соединение как файл, который Вы можете читать, и в который Вы можете записывать данные. Единственная особенность, файл может обладать определенным интеллектом и обрабатывать передаваемые Вами команды. Эти функции обеспечиваются расширением программы сетевой библиотеки `Javajava.net.*`;

Сокеты

Сокет(гнездо, разъем) (*сокет*=*IP*+*ПОРТ*) - это программная абстракция, используемая для представления “терминалов” соединений между двумя машинами. Для такого соединения, существует сокет на каждой машине, и можно представить себе виртуальный “кабель” соединяющий две машины, каждый конец которого вставлен в сокет. В Java сокет создается для установления соединения с другой машиной. Для этого в JAVA используется два класса сокетов, основанных на потоках: *ServerSocket* – класс используемый сервером, чтобы “слушать” входящие соединения и *Socket*-используемый клиентом для инициирования соединения. Как только клиент создает соединение по сокету, *ServerSocket* возвращает ему с помощью метода *accept()* соответствующий объект *Socket* на сервере, по которому будет происходить связь со стороны сервера. Начиная с этого момента, появляется соединение *Socket* к *Socket*, и можно считать эти соединения одинаковыми, т.к. они действительно одинаковые.

В распоряжение получаем 2 класса *InputStream* и *OutputStream* (либо с помощью соответствующих преобразователей, *Reader* и *Writer*) из сокета, который соответствующим образом представляет соединение, как потоковый объект ввода вывода. Можно использовать методы *getInputStream()* и *getOutputStream()* для создания соответствующих объектов *InputStream* и *OutputStream*.

Сокеты TCP/IP клиентов

Для создания сокета клиента используется конструктор:

`Socket(String hostname, int port)`// создает сокет, соединяющий локальную хост-машину с именованной хост-машиной и портом; может выбрасывать исключение `UnknownHostException` или `IOException`

`Socket(InetAddress address, int port)`// создает сокет, аналогичный предыдущему, но используется уже существующий объект класса `InetAddress` и порт; может выбрасывать исключение `IOException`

Сокет может в любое время просматривать связанную с ним адресную и портовую информацию при помощи методов, представленных в табл. 1.

Таблица 1

Методы просмотра адресной и портовой информации

Метод	Описание
InetAddress.getInetAddress()	Возвращает InetAddress-объект, связанный с Socket-объектом
int.getPort()	Возвращает удаленный порт, с которым соединен данный Socket-объект
int.getLocalPort()	Возвращает локальный порт, с которым соединен данный Socket-объект

После создания Socket-объект можно применять для получения доступа к связанным с ним потокам ввода/вывода. Данные потоки используются точно также как потоки ввода/вывода, описанные в лабораторной 1.

Пример кода клиента показан ниже.

Сокеты TCP/IP серверов

Как было указано выше для создания сокетов серверов используется класс ServerSocket. Указанный класс используется для создания серверов, которые прослушивают либо локальные, либо удаленные программы клиента, чтобы соединяться с ними на опубликованных портах.

Конструкторы класса ServerSocket:

ServerSocket(intport);// создает сокет сервера на указанном порте с длиной очереди по умолчанию 50

ServerSocket(intport, intmaxQueue);// создает сокет сервера на указанном порте с максимальной длиной очереди maxQueue

ServerSocket(intport, intmaxQueue, InetAddresslocalAddress);// создает сокет сервера на указанном порте с максимальной длиной очереди maxQueue

Класс ServerSocket имеет дополнительный метод accept(), который является блокирующим вызовом: он будет ждать клиента, чтобы инициализировать связь, и затем вернет нормальный Socket-объект, который будет использоваться для связи с клиентом.

После создания объекта ServerSocket его можно также применять для получения доступа к связанным с ним потокам ввода/вывода. Данные потоки используются точно также как потоки ввода/вывода, описанные в лабораторной 1.

Пример кода сервера показан ниже.

Рассмотрим пример. Необходимо создать приложение клиент-сервер, в котором клиент считывает строку с клавиатуры, отображает ее на экране, передает ее серверу, сервер отображает ее на экране, переводит в верхний регистр и передает клиенту, который, в свою очередь, снова отображает ее на экране.

Листинг кода

Программа сервера:

```
import java.io.*; // импорт каталога, содержащего классы для
ввода/вывода
import java.net.*; // импорт каталога, содержащего классы для работы в
сети Интернет
public class server { // описание класса server
    public static void main(String[] arg) { // создание метода main, без которого
Java-интерпретатор не запустит класс, с входным параметром с именем args,
который является массивом объектов - представителей класса String.
        ServerSocket serverSocket = null; // объявляется объект класса
ServerSocket
        Socket clientAccepted = null; // объявляется объект класса Socket
        ObjectInputStream sois = null; // объявляется байтовый поток ввода
        ObjectOutputStream soos = null; // объявляется байтовый поток вывода
        try { // ключевое слово механизма обработки исключений
            System.out.println("server starting..."); // вывод строки на консоль
            serverSocket = new ServerSocket(2525); // создание сокета сервера для
// заданного порта
            clientAccepted = serverSocket.accept(); // выполнение метода, которое
// обеспечивает реальное подключение сервера к клиенту
            System.out.println("connection established..."); // вывод строки на
// консоль
            sois =
new ObjectInputStream(clientAccepted.getInputStream()); // создание потока
ввода
            soos =
new ObjectOutputStream(clientAccepted.getOutputStream()); // создание потока
вывода
            String clientMessageReceived = (String) sois.readObject(); // объявляется
// строка и ей присваиваются данные потока ввода, представленные в виде
строки // (передано клиентом)
```

```

        while(!clientMessageRecieved.equals("quite"))//выполнение цикла: пока
//строка не будет равна "quite"
        {
            System.out.println("messagerecieved:
            "+clientMessageRecieved+"");//выведение на консоль строки и значения
            строковой //переменной
            clientMessageRecieved =
            clientMessageRecieved.toUpperCase();//приведение символов строки к
            верхнему реестру
            soos.writeObject(clientMessageRecieved);//потоку          вывода
            //присваивается значение строковой переменной (передается клиенту)
            clientMessageRecieved = (String)sois.readObject();//строке
            //присваиваются данные потока ввода, представленные в виде строки
            (передано клиентом)
        }
    }
    catch(Exception e) { //обработка исключения типа Exception
    }
    finally { //исключение передается умалчиваемому обработчику
    try { //ключевое слово механизма обработки исключений
    sois.close();//закрывается поток ввода
    soos.close();//закрывается поток вывода
    clientAccepted.close();//закрывается сокет
    serverSocket.close();//происходит отключение сервера от //клиента
    } catch(Exception e) { //обработка исключения типа Exception
    e.printStackTrace();//вызывается метод исключения e
    }
    }
    }
    }
    }

```

Программа клиента:

```

import java.io.*; //импорт каталога, содержащего классы для
ввода/вывода
import java.net.*; //импорт каталога, содержащего классы для работы в
сети Интернет
public class client { //описание класса client

```

```

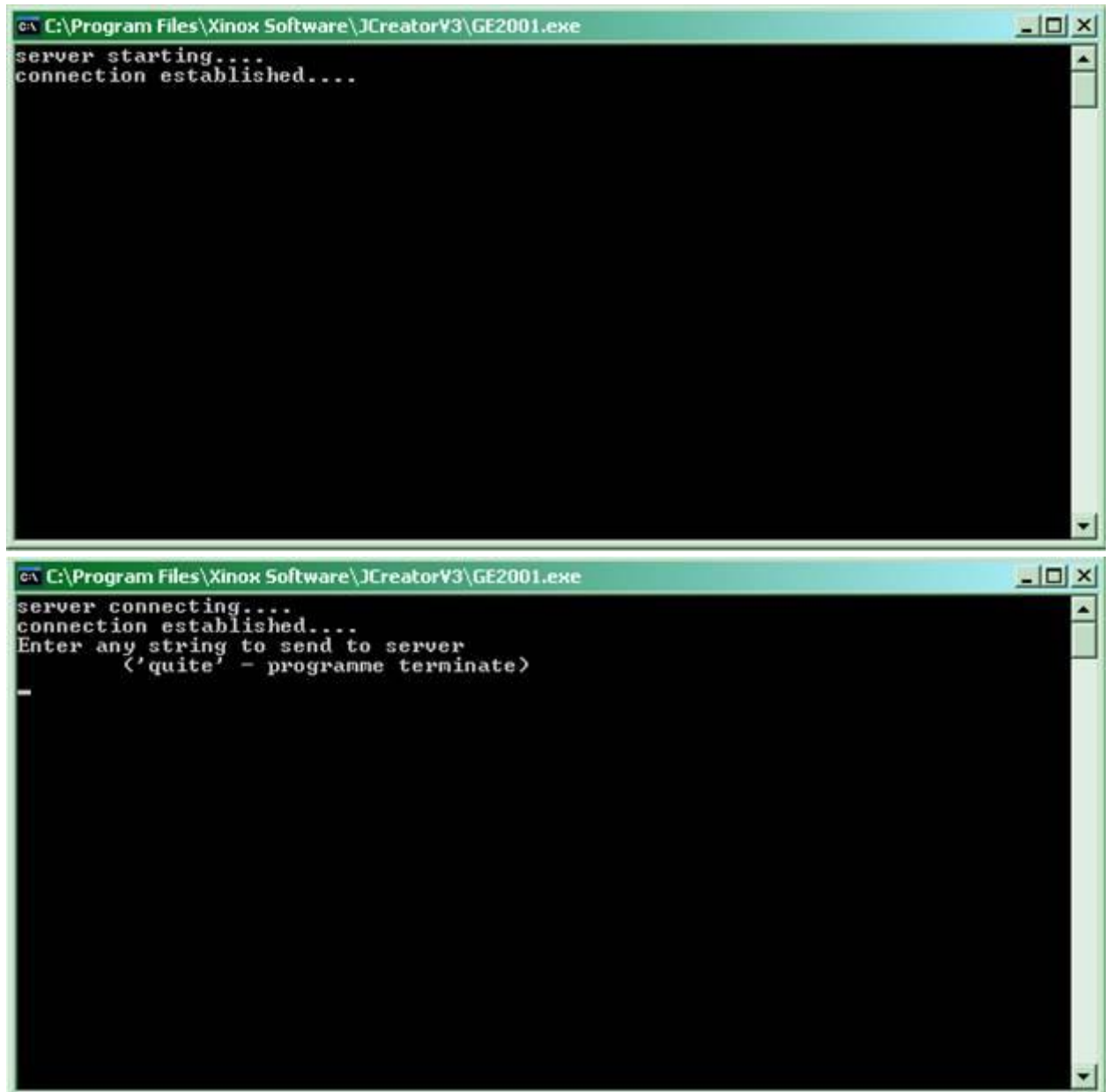
    public static void main(String[] arg) { //создание метода main, без которого
Java-//интерпретатор не запустит класс, с входным параметром с именем args,
который //является массивом объектов - представителей класса String.
        try { //ключевое слово механизма обработки исключений
            System.out.println("serverconnecting....");//вывод строки на консоль
            Socket clientSocket = new Socket("127.0.0.1",2525);//установление
//соединения между локальной машиной и указанным портом узла Интернет
            System.out.println("connectionestablished....");//вывод строки на
//консоль
            BufferedReader stdin =
new BufferedReader(new InputStreamReader(System.in));//создание
буферизированного символьного потока ввода
            ObjectOutputStream coos =
new ObjectOutputStream(clientSocket.getOutputStream());//создание потока
вывода
            ObjectInputStream cois =
new ObjectInputStream(clientSocket.getInputStream());//создание потока ввода
            System.out.println("Enter any string to send to server\n\t('quite'
-
programm to terminate)");//вывод строки на консоль
            String clientMessage = stdin.readLine();//ввод текста с клавиатуры
            System.out.println("you've entered: "+clientMessage);//вывод на консоль
//строки и значения строковой переменной
            while(!clientMessage.equals("quite")) { //выполнение цикла, пока строка
//не будет равна "quite"
                coos.writeObject(clientMessage);//поток вывода присваивается
//значение строковой переменной (передается серверу)
                System.out.println("~server~: "+cois.readObject());//выводится на //экран
содержимое потока ввода (переданное сервером)
                .out.println("-----");//выводится на //консоль
                clientMessage = stdin.readLine();//ввод текста с клавиатуры
                System.out.println("you've entered: "+clientMessage);//вывод на //консоль
строки и значения строкой переменной
            }
            coos.close();//закрытие потока вывода
            cois.close();//закрытие потока ввода
            clientSocket.close();//закрытие сокетного соединения
        } catch (Exception e) { //обработка исключения типа Exception
            e.printStackTrace();//выполнение метода исключения e
        }
    }

```

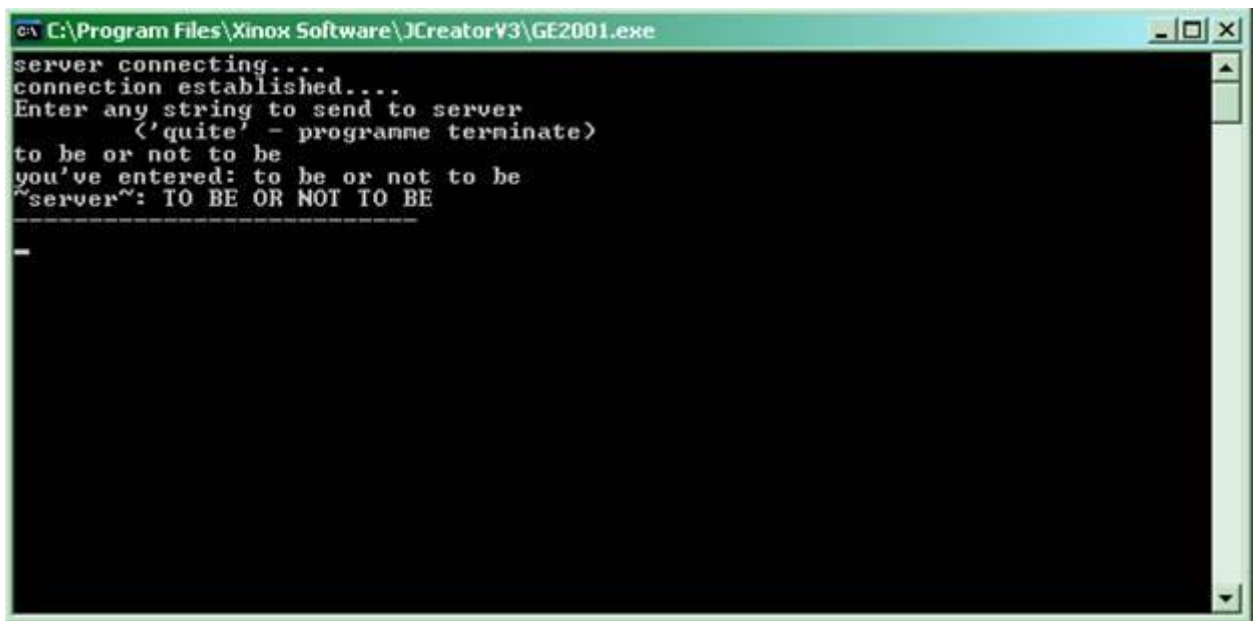
```
}  
}
```

Для того чтобы программа заработала, надо оба исходных текста с расширением .java открыть, скажем, в JCreator'е, причем в одном окне, откомпилировать и запустить сначала сервер, затем клиент.

В результате проделанной работы на экране появится консольное окно сервера и клиента:



Теперь введем в окно клиента какую-либо строку:



```
C:\Program Files\Xinox Software\JCreatorV3\GE2001.exe
server connecting....
connection established....
Enter any string to send to server
    ('quite' - programme terminate)
to be or not to be
you've entered: to be or not to be
^server~: TO BE OR NOT TO BE
-

```



```
C:\Program Files\Xinox Software\JCreatorV3\GE2001.exe
server starting....
connection established....
message recieved: 'to be or not to be'
-

```

Сначала строка отразилась в клиентском окне, затем сервер вывел полученную им строку в своем окне, затем клиент вывел строку, переданную сервером, и ожидает ввода новой строки. Теперь введем слово «quite». Программа завершила свою работу. Сначала закрылось окно клиента, затем – окно сервера.

ЗАДАНИЯ

Разработать приложение на основе ТСП-соединения, позволяющее осуществлять взаимодействие клиента и сервера по совместному решению задач обработки информации. Приложение должно располагать возможностью передачи и модифицирования получаемых (передаваемых) данных.

Варианты заданий Части № 1 выбираются по последней цифре номера зачетки (при 0 выбирается 10 вариант).

1. Разработать приложение-калькулятор для совершения простейших арифметических операций. Исходные параметры и тип операции (+,-,/,*) вводятся на клиентской части и передаются серверу. Сервер возвращает клиенту результат операции.

2. Разработать приложение-чат. На сервере и клиенте отображаются передаваемые сообщения и время их отправления.

3. Разработать приложение-генератор случайных чисел. На клиентской части вводится целое положительное число N и передается серверу, а тот в свою очередь возвращает клиенту массив случайных чисел от 1 до N.

4. Разработать приложение-поисковик слов. На сервере хранится определенный текст. На клиентской части вводится слово для поиска и передается серверу, а тот в свою очередь осуществляет поиск этого слова в тексте и возвращает клиенту все предложения, в которых встречается это слово.

5. Разработать приложение-счетчик букв. На клиентской части вводится строка и передается серверу, а тот в свою очередь осуществляет подсчет гласных и согласных букв и возвращает этот результат клиенту.

6. Разработать приложение-определитель матрицы. На клиентской части вводится исходная матрица произвольного порядка и передается серверу, а тот в свою очередь вычисляет определитель этой матрицы и возвращает результат клиенту.

7. Разработать приложение для нахождения обратной матрицы размером 3*3. Исходная матрица вводится на клиентской части и передается серверу, а тот в свою очередь возвращает клиенту обратную матрицу.

8. Разработать приложение для определения счастливого лотереи. На сервере хранятся номера билетов. На каждом билете имеются 10 случайных чисел от 1 до 100. На клиентской части вводятся 10 чисел от 1 до 100 и сервер должен определить номер билета, в котором имеется больше всего совпадений с введенными числами.

9. Разработать приложение для определения призовых мест на соревнованиях по прыжкам в длину. На сервере хранятся фамилии участников соревнований и их идентификационные номера. На клиентской части вводятся результаты прыжков по каждому идентификационному номеру, а сервер возвращает фамилии спортсменов, занявших 1, 2 и 3 места.

10. Разработать приложение для определения суммы подоходного налога. На клиентской части вводятся заработные платы сотрудников предприятия и передаются серверу, а тот в свою очередь возвращает суммы налога. Причем для з/п меньше 100.000 руб. применяется ставка налога 5%, для з/п от 100.000 до 500.000 – ставка 10%, для з/п больше 500.000 – ставка 15%.

ЧАСТЬ 2

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Разработка консольных приложений с использованием сокетов (разработка UDP соединений)

Цель: Разработка клиент-серверных приложений с использованием протокола UDP

Скрыть объявление

Протокол UDP

UDP представляет собой альтернативу TCP, требующую меньших накладных расходов. В отличие от TCP, UDP имеет следующие характеристики:

- Ненадежный (unreliable). UDP не имеет ни встроенного механизма обнаружения ошибок, ни средств повторной пересылки поврежденных или потерянных данных.
- Без установления логического соединения (connectionless). Перед пересылкой данных UDP не устанавливает логического соединения. Информация пересылается в предположении, что принимающая сторона ее ожидает.
- Основанный на сообщениях (message-oriented). UDP позволяет приложениям пересылать информацию в виде сообщений, передаваемых посредством дейтаграмм (datagram), которые являются единицами передачи данных в UDP. Приложение должно самостоятельно распределить данные по отдельным дейтаграммам.

Для некоторых приложений UDP подходит лучше, чем TCP. Например, в протоколе NTP (NetworkTimeProtocol — протокол передачи времени) потерянный пакет с информацией о текущем времени к моменту повторной передачи содержал бы неверные данные. Сетевая файловая система (NetworkFileSystem, NFS) более эффективно обеспечивает надежность на уровне приложения, и потому использует UDP.

Как и в TCP, в UDP применяется схема адресации с использованием портов, позволяющая нескольким приложениям параллельно принимать и

посылать данные. В то же время порты UDP отличаются от портов TCP. Например, одно приложение может отзывать на номер 512 порта UDP, а при этом другой независимый сервис может обрабатывать порт 512, относящийся к TCP.

Программирование с использованием UDP имеет несколько важных областей применения. Для правильного использования необходимо знать основные характеристики этого протокола.

UDP хорошо применим для приложений, которые обмениваются данными в виде отдельных сообщений, т. е. один запрос от клиента порождает один ответ от сервера. Для UDP очень хорошо подходят данные, зависящие от времени. По сравнению с TCP протокол UDP требует значительно меньших накладных расходов, зато вынуждает разработчика самостоятельно обеспечивать требуемую надежность. Например, если клиент не получает ответа на свои запросы, что вполне вероятно при использовании UDP, возможно, в программе следует предусмотреть повторение запроса либо вывод сообщения, предупреждающего о ненадежной связи.

UDP существенно отличается от TCP. UDP является ненадежным, основанным на сообщениях протоколом без установления логического соединения. Наиболее подходящая для UDP аналогия — связь посредством почтовых открыток.

В протоколе UDP диалог должен быть разделен на небольшие сообщения, которые укладываются в небольшой пакет определенного размера. Когда посылается сообщение, нельзя быть уверенным, что ответ будет получен: сообщение могло быть потеряно по пути, мог потеряться ответ получателя, получатель также мог игнорировать сообщение.

Почтовые открытки, которыми обмениваются сетевые программы, называются дейтаграммами (datagrams). Дейтаграмма содержит массив байтов. Принимающая программа может извлечь этот массив и декодировать информацию, а затем, возможно, послать ответную дейтаграмму.

Как и для протокола TCP, программирование для UDP будет использовать абстракцию сокета, но сокеты UDP сильно отличаются от сокетов TCP. Если продолжить почтовую аналогию, то сокет UDP соответствует почтовому ящику.

Почтовый ящик идентифицируется адресом владельца, но нам не придется заводить новый ящик для каждого, кому мы будем посылать сообщения (можно, однако, создать отдельный ящик для газет, чтобы они не попадали в ящик для писем). Для отправки сообщения достаточно

написать на открытке адрес, по которому она должна быть доставлена. Затем она помещается в почтовый ящик и (раньше или позже) уходит по назначению.

Можно, в принципе, бесконечно долго ожидать, пока сообщение дойдет до почтового ящика. Когда сообщение получено, его можно прочесть. На открытке содержится также метайнформация, позволяющая по обратному адресу получить сведения об отправителе сообщения.

Итак, программирование с использованием UDP требует решить следующие задачи:

- Создание правильно адресованной дейтаграммы
- Создание сокета для рассылки и получения дейтаграмм данным приложением
- Помещение дейтаграмм в сокет для передачи по назначению
- Ожидание получения дейтаграмм из сокета
- Декодирование дейтаграмм для выделения самого сообщения, адреса отправителя и другой метайнформации

Классы UDP

Необходимые средства поддержки протокола UDP находятся в пакете `java.net`. Для создания дейтаграмм в Java существует класс `DatagramPacket`. При получении дейтаграммы по протоколу UDP класс `DatagramPacket` используется также для чтения данных, адреса отправителя и метайнформации.

Чтобы создать дейтаграмму для отправки на удаленную машину, используется следующий конструктор:

```
public DatagramPacket(byte[] ibuf, intlength, InetAddressiaddr, intiport);
```

Здесь `ibuf` — массив байт, содержащий кодированное сообщение; `length` — количество байт, которое должно быть помещено в пакет, что определяет размер дейтаграммы; `iaddr` — это экземпляр класса `InetAddress`, который хранит IP-адрес получателя; `iport` указывает номер порта, на который посылается дейтаграмма.

Чтобы получить дейтаграмму, необходимо использовать другой конструктор для объекта `DatagramPacket`, в котором будут находиться принятые данные. Прототип конструктора имеет вид:

```
public DatagramPacket(byte[] ibuf, intlength);
```

Здесь `ibuf` — массив байт, куда должны быть скопированы данные из дейтаграммы, а `length` — количество байт, которое должно быть скопировано.

Дейтаграммы не ограничены определенной длиной; можно создавать как очень длинные, так и очень короткие дейтаграммы. Заметим, однако,

что между клиентом и сервером должно существовать соглашение о длине дейтаграмм, поскольку оба они должны создать массив байт нужного размера перед созданием объекта `DatagramPacket` для отправки или получения дейтаграммы.

Когда дейтаграмма получена, как будет продемонстрировано ниже, можно прочитать ее данные. Другие методы позволяют получить метаданные, относящиеся к сообщению:

Программирование для Учимдетей думать
самостоятельно и раскрывать свой потенциал! Выгодные цены! Учимдетей
думать самостоятельно и раскрывать свой потенциал! Выгодные
цены!coddyschool.comО насБольшой выбор
курсовСистемаскидокКонтактыПерейтиЯндекс.ДиректСкрыть объявление

```
§ public int getLength();  
§ public byte[] getData();  
§ public InetAddress getAddress();  
§ public int getPort();
```

Метод `getLength()` возвращает количество байт, из которых состоят данные дейтаграммы. Метод `getData()` позволяет получить массив, содержащий эти данные. `getAddress()` возвращает адрес отправителя, а `getPort()` — номер порта UDP, используемый отправителем.

Отправка и получение дейтаграмм осуществляется при помощи класса `DatagramSocket`, который создает сокет UDP. У него есть два конструктора, один из которых позволяет системе назначить любой из свободных портов UDP. Другой дает возможность задать конкретный порт, что полезно при разработке сервера. Как и для портов TCP, в большинстве операционных систем порты с номерами меньше 1024 доступны только процессам с привилегиями суперпользователя.

```
public DatagramSocket() throws SocketException;  
public DatagramSocket(int port) throws SocketException;
```

Сокет, созданный первым конструктором, можно использовать для отправки правильно адресованных дейтаграмм при помощи следующего метода класса `DatagramSocket`:

```
public void send(DatagramPacket p) throws IOException;
```

Если `DatagramSocket` создан вторым конструктором, можно получить дейтаграмму:

```
public synchronized void receive(DatagramPacket p) throws IOException;
```

Заметим, что метод `receive()` блокируется до момента получения дейтаграммы. Поскольку UDP является ненадежным протоколом, нельзя быть уверенным, что возврат из `receive()` вообще произойдет.

Когда закончен обмен через сокет UDP, его следует закрыть методом

```
public synchronized void close();
```

авайте начнем вместе с сервера. Метод *main()* создает образец сервера и вызывает метод *runServer()*. Этот метод связывается с портом 8001 на локальной главной ЭВМ и ждет пакет. После того, как пакет приходит, он извлекает команду и посылает ответ обратно отправителю (клиенту). Если это была команда выхода, то сервер закрывает себя, а также закрывает сокет, на котором осуществлялось прослушивание. Обратите внимание, что при создании датаграмного пакета для посылки, мы также внедряем адрес адресата и порта.

Клиент начинает работу, соединяясь с сервером на локальной главной ЭВМ на порте 8001, а затем посылает команду для выбора версии сервера. Затем используется класс *java.net.InetAddress* для определения адреса сервера. После этого он отображает полученный ответ. В заключение, осуществляется посылка команды для выключения сервера и закрытия сокета.

Листинг UDPServer.java

```
import java.net.*;
import java.io.*;

/** The UDP server. */
public class UDPServer {

    /** The default port for the server. */
    public final static int DEFAULT_PORT = 8001;
    /** The version command. */
    public final String VERSION_CMD = "VERS";
    /** The quit command. */
    public final String QUIT_CMD = "QUIT";
    /** The server version. */
    public final byte[] VERSION = { 'V', '2', '.', '0' };
    /** The unknown command. */
    public final byte[] UNKNOWN_CMD = { 'U', 'n', 'k', 'n', 'o', 'w', 'n', ' ',
        'c', 'o', 'm', 'm', 'a', 'n', 'd' };
    /** Run the server. */
    public void runServer() throws IOException {
```

```

DatagramSocket s = null;
try {
// Thestopflag
booleanstopFlag = false;
// Thebuffertohold a datagrampacket
byte[] buf = newbyte[512];
// Createthesocketandbindto a particularport
s = newDatagramSocket(DEFAULT_PORT);
System.out.println("UDPServer: Startedon "
+ s.getLocalAddress() + ":"
+ s.getLocalPort());
while(!stopFlag) {
// Receive a packet
DatagramPacketrecvPacket = newDatagramPacket(buf, buf.length);
s.receive(recvPacket);
// Extractthecommandfromthepacket
Stringcmd = newString(recvPacket.getData()).trim();
System.out.println("UDPServer: Command: " + cmd);
// Theresponepacket
DatagramPacketsendPacket = newDatagramPacket(buf, 0,
recvPacket.getAddress(),
recvPacket.getPort());
// Thenumberofbytesintheresponse
int n = 0;
// Handlethecommand
if (cmd.equals(VERSION_CMD)) {
n = VERSION.length;
System.arraycopy(VERSION, 0, buf, 0, n);
}
elseif (cmd.equals(QUIT_CMD)) {
// Stoptheserver
stopFlag = true;
continue;
}
else {

```

```

n = UNKNOWN_CMD.length;
System.arraycopy(UNKNOWN_CMD, 0, buf, 0, n);
}
// Updatetheresponse
sendPacket.setData(buf);
sendPacket.setLength(n);
// Sendtheresponse
s.send(sendPacket);
} // while(serverisnotstopped)
System.out.println("UDP Server: Stopped");
}
finally {
// Close
if (s != null) {
s.close();
}
}
}
}
/** Main. */
public static void main(String[] args) {
try {
UDPServerudpSvr = newUDPServer();
udpSvr.runServer();
}
catch(IOExceptionex) {
ex.printStackTrace();
}
}
}
//Листинг UDPClient.java
importjava.net.*;
importjava.io.*;
/** The UDP client. */
publicclassUDPClient {
/** Runstheclient. */
publicvoidrunClient() throwsIOException {
DatagramSocket s = null;
try {
// Thebuffer

```

```

byte[] buf = newbyte[512];
// Create a socketonanyavailableport
s = newDatagramSocket();
System.out.println("UDPClient: Started");
// Theversioncommand
byte[] verCmd = { 'V', 'E', 'R', 'S' };
// Thedatatotransmit
DatagramPacket sendPacket =
newDatagramPacket(verCmd, verCmd.length,
InetAddress.getByName("127.0.0.1"), 8001);
s.send(sendPacket);
// Fetchtheserverresponse
DatagramPacket recvPacket = newDatagramPacket(buf, buf.length);
s.receive(recvPacket);
// Extracttheserverversion
String version = newString(recvPacket.getData()).trim();
System.out.println("UDPClient: ServerVersion: " + version);
// Thequitcommand
byte[] quitCmd = { 'Q', 'U', 'I', 'T' };
// Shutdowntheserver
sendPacket.setData(quitCmd);
sendPacket.setLength(quitCmd.length);
s.send(sendPacket);
System.out.println("UDPClient: Ended");
}
finally {
// Close
if (s != null) {
s.close();
}
}
}
}
/** Main. */
publicstaticvoidmain(String[] args) {
try {
UDPClientclient = newUDPClient();
client.runClient();
}
catch(IOExceptionex) {

```

```

ex.printStackTrace();
}
}
}

```

Вначале запустите сервер, затем клиент. Результаты работы приложений:

Организуем сервер, который в ответ на запрос клиента (полученный пакет (датаграмму)) будет возвращать ему пакет с текущей датой и временем, который распечатывается на компьютере клиента.

Серверная часть приложения:

```

import java.net.*;
import java.util.*;
public class UDPServer { //заголовок класса
public UDPServer() { //конструктор
try {
System.out.println("Receiving packets..."); //вывод сообщения
DatagramSocket ds = new DatagramSocket(61); //номер прослушиваемого
порта
while (true) { // бесконечный цикл
Date date;
byte data[] = new byte[6]; //организация массива под дату
DatagramPacket dp = new DatagramPacket(data, 6); //заготовка пакета
ds.receive(dp); //ожидание обращения
int clientPort = dp.getPort(); //получить номер порта клиента
InetAddress clientAddress = dp.getAddress(); //получить адрес узла
клиента
date = new Date(); //организовать объект
data[0] = (byte) date.getDate(); //получить число, месяц, год, мин., сек.
data[1] = (byte) date.getMonth();
data[2] = (byte) date.getYear();
data[3] = (byte) date.getHours();
data[4] = (byte) date.getMinutes();
data[5] = (byte) date.getSeconds();
//организовать пакет для клиента
dp = new DatagramPacket(data, 6, clientAddress, clientPort);
//выдать сообщение
System.out.println("Sending packets...");
ds.send(dp); //отослать пакет клиенту

```

```

System.err.println("After"); //сообщить
}
}
catch(Exceptionexc) { //обработка исключений
exc.printStackTrace();
}
}
publicstaticvoidmain(String[] args) {
UDPServerudpServer = newUDPServer(); //создать объект udpServer }}

```

Клиентская часть приложения:

```

importjava.net.*;
publicclassUDPClient {
publicUDPClient(Stringserver) {
try {
DatagramSocketds = newDatagramSocket();
bytedata[]=newbyte[6];
DatagramPacketdp=newDatagramPacket(data,
InetAddress.getByName(server), 61);
ds.send(dp);
System.out.println("Waitingfordata...");
dp=newDatagramPacket(data, 6);
ds.receive(dp);
System.out.println("Data'sreceived.");
//формирование даты в удобочитаемом виде и распечатка даты
System.out.print("Date:"+data[0]+". "+(data[1]+1)+". "+(data[2]+1900)+"
Time:"+data[3]+". "+data[4]+". "+data[5]); }
catch(Exceptionexc) { //обработка исключительных ситуаций
exc.printStackTrace();
} }
publicstaticvoidmain(String[] args) { //при запуске указать доменное
имя
UDPClientudpClient = newUDPClient(args[0]); //сервера
} }

```

ЗАДАНИЯ

Разработать приложение на основе UDP-соединения, позволяющее осуществлять взаимодействие клиента и сервера по совместному решению задач обработки информации. Приложение должно располагать

возможностью передачи и модифицирования получаемых (передаваемых) данных.

Возможности клиента:

– передать серверу исходные параметры (вводятся с клавиатуры) для расчета значения функции;

– получить расчетное значение функции

Возможности сервера:

– по полученным от клиента исходным параметрам рассчитать значение функции;

– передать клиенту расчетное значение функции;

– сохранить исходные параметры и значение функции в файл.

Варианты заданий части № 2 выбираются по сумме двух последних цифр номера договора.

$$1. Y = (\pi x + \arctg|x| -$$

$$2. Y = \frac{bx}{|x|} + \frac{x}{\log_2}$$

$$3. Y = \frac{\lg|a^7| + -}{e}$$

$$4. Y = (\arctg x^2 + \cos \sqrt{s}$$

$$5. Y = e^x + \ln|x^{1/4}|$$

$$6. Y = \sqrt[3]{\frac{ax+b}{cx-d}}$$

$$7. Y = \frac{\lg|x| - \ln(2)}{\arctg(x) +}$$

$$8. Y = (x^{1/3} + \operatorname{sign}(x$$

$$9. Y = \ln|x^{\cos(x)}| + i$$

$$10. Y = \log_2(x + \cos \sqrt{|\sin$$

$$11. Y = (a + b - ct$$

$$12. Y = b + e^{\sqrt{|x + \cos$$

$$13. Y = \frac{3x^2 + 15x^3}{|x^{10}| + \ln($$

14. $Y=\sqrt{tg^2(x)+lg}$

15. $Y=\arcsin(\cos x+\gamma$

16. $\alpha=\ln(y^{-\sqrt{|x|}})*(x-\frac{y}{2})+\sin^2ar$

17. $\beta=\sqrt{10(\sqrt{x}+x^{yz})(\sin^2z$

18. $\chi=5arctgx-\frac{1}{4}\cos^2$

19. $\varphi=\frac{e^{|x-y|}|x-y|^{x+y}}{arctgx+tgz}+\gamma$

20. $\phi=\left|x^{\frac{y}{x}}-\sqrt{\frac{y}{x}}\right|+(y-x)$

21. $\delta=y^x+\sqrt{|x|+e^y}$

22. $\varepsilon=\left|\cos x-e^y\right|^{1+2\ln^2y}(1+$

23. $\lambda=\frac{1+\sin^2(x+y)}{\left|e^x-\frac{2y}{1+x^2y^3}\right|}x^{|y|}+\cos$

24. $\eta=\frac{\sqrt{8+|x+y|^2+z}}{x^2+y^2+z^2}-e^{|x-y|}(t,$

25. $\sigma=\frac{2\cos(x-\frac{\pi}{6})}{e^{0.5}+\sin^2y}(1-$

$$26. \quad \chi = y + \frac{e^{x-y}}{y + \frac{x^2}{y + \frac{x^3}{y}}}$$

$$27. \quad \theta = \lg(\sqrt{e^{x-y} + x^{|y|}} + z)(x$$

$$28. \quad \varpi = 6 + \frac{e^{x-\sin y}}{y + \frac{\lg(x^2)}{(y + \frac{x^7}{z})}}(1 + \epsilon$$

$$29. \quad \tau = \frac{2x(1+x^2)^2}{x + \sqrt[3]{1+x^5}}$$

$$30. \quad \xi = \frac{x + \frac{y}{5 + \sqrt{x}}}{|y-x| + \sqrt[3]{x}} e$$

3. ОБОРУДОВАНИЕ

ПЭВМ IBMPC, SVGA монитор с разрешением не менее 800*600 пикселей; клавиатура; мышь. Среда разработки Java.

4. СОДЕРЖАНИЕ ОТЧЕТА ПО КРЗ

1. Цель и задачи работы
 2. Блок-схемы и тексты программ
- Результаты и выводы работе