**Работа в сети.**

**1) Утилита telnet:**

Утилита telnet служит отличным инструментальным средством для отладки сетевых программ. Она должна запускаться из командной строки по команде telnet. Ниже приводится один из примеров необычного использования этой утилиты:

* telnet time-a.nist.gov 13

Когда сеанс связи с сервером через порт 13 начинается по команде telnet с параметром time-a.nist.gov, сетевое программное обеспечение преобразует строку "time-a.nist.gov" в IP-адрес 129.6.15.28. Затем оно посылает по этому адресу запрос на соединение с удаленным компьютером через порт 13. После установления соединения программа на удаленном компьютере передает обратно строку с данными, а затем разрывает соединение.

**2) Подключение через сокет:**

В данной программе наибольший интерес представляют следующие две строки кода:

* Socket s = new Socket("time-a.nist.gov", 13);
* Scanner in = new Scanner(s.getlnputStream(), "UTF-8"))

В первой строке кода открывается сокет. Сокет — это абстрактное понятие, обозначающее возможность для программ устанавливать соединения для обмена данными по сети. Конструктору объекта сокета передается адрес удаленного сервера и номер порта. Если установить соединение не удается, генерируется исключение типа UnknownHostException, а при возникновении каких-нибудь других затруднений — исключение типа IOException. Класс UnknownHostException является подклассом, производным от класса IOException, поэтому в данном простом примере обрабатывается только исключение из суперкласса.

После открытия сокета метод getlnputStream() из класса java.net. Socket возвращает объект типа InputStream, который можно использовать как любой другой поток ввода.

**3) Время ожидания:**

Для конкретной прикладной программы можно самостоятельно определить наиболее подходящую величину времени ожидания для сокета, а затем вызвать метод setSoTimeout (), чтобы установить эту величину в миллисекундах. В приведенном ниже фрагменте кода показано, как это делается.

* Socket s = new Socket(. . .);
* s.setSoTimeout(10000); // истечение времени ожидания через 10 секунд

Если величина времени ожидания была задана для сокета, то при выполнении всех последующих операций чтения и записи данных будет генерироваться исключение типа SocketTimeoutException (InterruptedlOException) по истечении времени ожидания до фактического завершения текущей операции.

Можно сначала создать несоединяемый сокет, а затем установить соединение с ним, задав время ожидания:

* Socket s = new Socket();
* s.connect(new InetSocketAddress(host, port), timeout);

**4) Сокеты сервера:**

4.1)Среди понятий и терминов, связанных с работой в сети, если одно очень важное – Сокет. Оно обозначает точку, через которую происходит соединение. Проще говоря, сокет соединяет в сети две программы.Так происходит общение в сети. Для начала вот возможные конструкторы класса **Socket:**

* Socket(String имя\_хоста, int порт) throws UnknownHostException, IOException

4.2)Допустим мы объявили, в виде класса **Socket**, на стороне клиента запрос на соединение. Как сервер разгадает наше желание? Для это сервер имеет такой класс как **ServerSocket**, и метод accept() в нём. Его конструкторы представлены ниже:

* ServerSocket() throws IOException
* ServerSocket(int порт) throws IOException
* ServerSocket(int порт, int максимум\_подключений) throws IOException

Создав объект типа **ServerSocket** необходимо выяснить, что с сервером кто-то хочет соединиться. Тут подключается метод accept(). Искомый ждёт пока кто-либо не захочет подсоединится к нему, и когда это происходит возвращает объект типа **Socket**, то есть воссозданный клиентский сокет. И вот когда сокет клиента создан на стороне сервера, можно начинать двухстороннее общение.

Создать объект типа **Socket** на стороне клиента и воссоздать его с помощью **ServerSocket** на стороне сервера – вот необходимый минимум для соединения.

4.3) Что мы уже имеем? Сокет с адресом сервера и номером порта у клиента, и тоже самое, благодаря accept(), на стороне сервера. Так что разумно предположить, что общаться они будут как раз через сокет. Для этого есть два метода которые дают доступ к потокам InputStream и OutputStream объекта типа **Socket**. Вот они:

* InputStream getInputStream()
* OutputStream getOutputStream()

Так как читать и писать голые байты не так эффективно - потоки можно обернуть в классы адаптеры, буферизированные, или нет. Например:

* BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
* BufferedWriter out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));

И да, не забывайте про метод flush() для **BufferedWriter** – он выталкивает содержимое буфера. Если вам это кажется неудобным, не расстраивайтесь, всегда можно воспользоваться классом **PrintWriter**, которым нужно обернуть out, указать вторым аргументом true и тогда выталкивание из буфера будет происходить автоматически:

* PrintWriter out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(socket.getOutputStream())), true);

Но является ли ввод/вывод строк пределом возможностей сокета? Нет, хотите оправлять объекты через потоки сокета? Ради бога. Сериализуйте их, и вперед:

* ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());
* ObjectInputStream in = new ObjectInputStream(socket.getInputStream());

**5) Многопоточность:**

В предыдущем простом примере серверной программы не предусмотрена возможность одновременного подключения сразу нескольких клиентских программ.Всякий раз, когда серверная программа устанавливает новое сокетное соединение, т.е. в результате вызова метода accept () возвращается сокет, запускается новый поток исполнения для подключения данного клиента к серверу. После этого происходит возврат в основную программу, которая переходит в режим ожидания следующего соединения. Для того чтобы все это произошло, в серверной программе следует организовать приведенный ниже основной цикл.

while (true) {

Socket incoming = s.acceptO;

Runnable r = new ThreadedEchoHandler(incoming);

Thread t = new Thread(r); t.start();

}

Класс ThreadedEchoHandler реализует интерфейс Runnable и в своем методе run () поддерживает взаимодействие с клиентской программой:

class ThreadedEchoHandler implements Runnable {

public void run() {

try (InputStream inStream = incoming.getInputStream();

OutputStream outStream = incoming.getOutputStream()) {

обработать полученный запрос и отправить ответ

} catch(IOException е) {

обработать исключение

}

Когда запускается новый поток исполнения, при каждом соединении, несколько клиентских программ могут одновременно подключаться к серверу. Это нетрудно проверить, выполнив следующие действия.

**6)Полузакрытие:**

Полузакрытые обеспечивает возможность прервать передачу данных на одной стороне сокетного соединения, продолжая в то же время прием данных от другой стороны. Рассмотрим типичную ситуацию.

**7) Прерываемые сокеты:**

Для прерывания сокетных операций служит класс SocketChannel, предоставляемый в пакете java.nio. Объект типа SocketChannel создается следующим образом:

* SocketChannel channel = SocketChannel.open(new InetSocketAddress(host, port));

У канала отсутствуют связанные с ним потоки ввода-вывода. Вместо этого в канале предоставляются методы read () и write (), использующие объекты типа Buffer. (Подробнее о буферах см. в главе 2.) Эти методы объявляются в интерфейсах ReadableByteChannel и Writable ByteChannel. Если же нет желания иметь дело с буферами, для чтения из канала типа SocketChannel можно воспользоваться объектом типа Scanner. Для этой цели в классе Scanner предусмотрен следующий конструктор с параметром типа ReadableByteChannel:

* Scanner in = new Scanner(channel, "UTF-8");

Чтобы превратить канал в поток вывода, применяется статический метод

* Channels.newOutputStream():
* OutputStream outStream = Channels.newOutputStream(channel);

Вот, собственно, и все, что нужно сделать для прерывания сокетной операции. Если же поток исполнения будет прерван в процессе установления соединения, чтения или записи, соответствующая операция завершится генерированием исключения.

**8) URL и URI:**

Объект типа URL создается следующим образом:

* URL url = new URL(символьная строка с URL);

Если требуется только извлечь содержимое из указанного ресурса, достаточно вызвать метод openStream() из класса URL. Этот метод возвращает объект типа InputStream. Поток ввода данного типа можно использовать обычным образом, например, создать объект типа Scanner:

* InputStream inStream = url.openStream();
* Scanner in = new Scanner(inStream, "UTF-8");

В пакете java.net отчетливо различаются унифицированные указатели ресурсов (URL) и унифицированные идентификаторы ресурсов (URI).

* URI – имя и адрес ресурса в сети, включает в себя URL и URN
* URL – адрес ресурса в сети, определяет местонахождение и способ обращения к нему
* URN – имя ресурса в сети, определяет только название ресурса, но не говорит как к нему подключиться

Рассмотрим примеры:

* URI – <https://wiki.merionet.ru/images/vse-chto-vam-nuzhno-znat-pro-devops/1.png>
* URL - [https://wiki.merionet.ru](https://wiki.merionet.ru/)
* URN - images/vse-chto-vam-nuzhno-znat-pro-devops/1.png

**9) Извлечение данных средствами класса URLConnection:**

Для получения дополнительных сведений о веб-ресурсе следует воспользоваться классом URLConnection, предоставляющим намного больше средств управления доступом к веб-ресурсам, чем более простой класс URL. Для работы с объектом типа URLConnection необходимо тщательно спланировать и выполнить следующие действия.

1. Вызвать метод openConnection () из класса URL для получения объекта типа URLConnection следующим образом:

* + - URLConnection connection = url.openConnection();

2. Задать свойства запроса с помощью перечисленных ниже методов.

* + - SetDoInput()
    - setDoOutput()
    - setlfModifiedSince()
    - setUseCaches()
    - setAllowUserlnteraction()
    - setRequestProperty()
    - setConnectTimeout()
    - setReadTimeout()

3. Эти методы будут подробно рассматриваться далее.

4. Установить соединение с удаленным ресурсом с помощью метода connect ():

* connection.connect();

5. Помимо создания сокета, для установления соединения с веб-сервером этот метод запрашивает также у сервера данные заголовка.

6. После подключения к веб-серверу становятся доступными поля заголовка. Обращаться к ним можно с помощью универсальных методов getHeaderFieldKey () и getHeaderField (). Кроме того, для удобства разработки предусмотрены перечисленные ниже методы обработки стандартных полей запроса.

* getContentType()
* getContentLength()
* getContentEncoding()
* getDate()
* getExpiration()
* getLastModified()

7. И наконец, для доступа к данным указанного ресурса следует вызвать метод getlnputStreamf), предоставляющий поток ввода для чтения данных. (Это тот же самый поток ввода, который возвращается методом openStream () из класса URL.) Существует также метод getContent (), но он не такой удобный. Для обработки содержимого стандартных типов, например, текста (text/plain) или изображений (image/gif), придется воспользоваться классами из пакета com. sun. Кроме того, можно зарегистрировать собственные обработчики содержимого, но они в данной книге не рассматриваются.