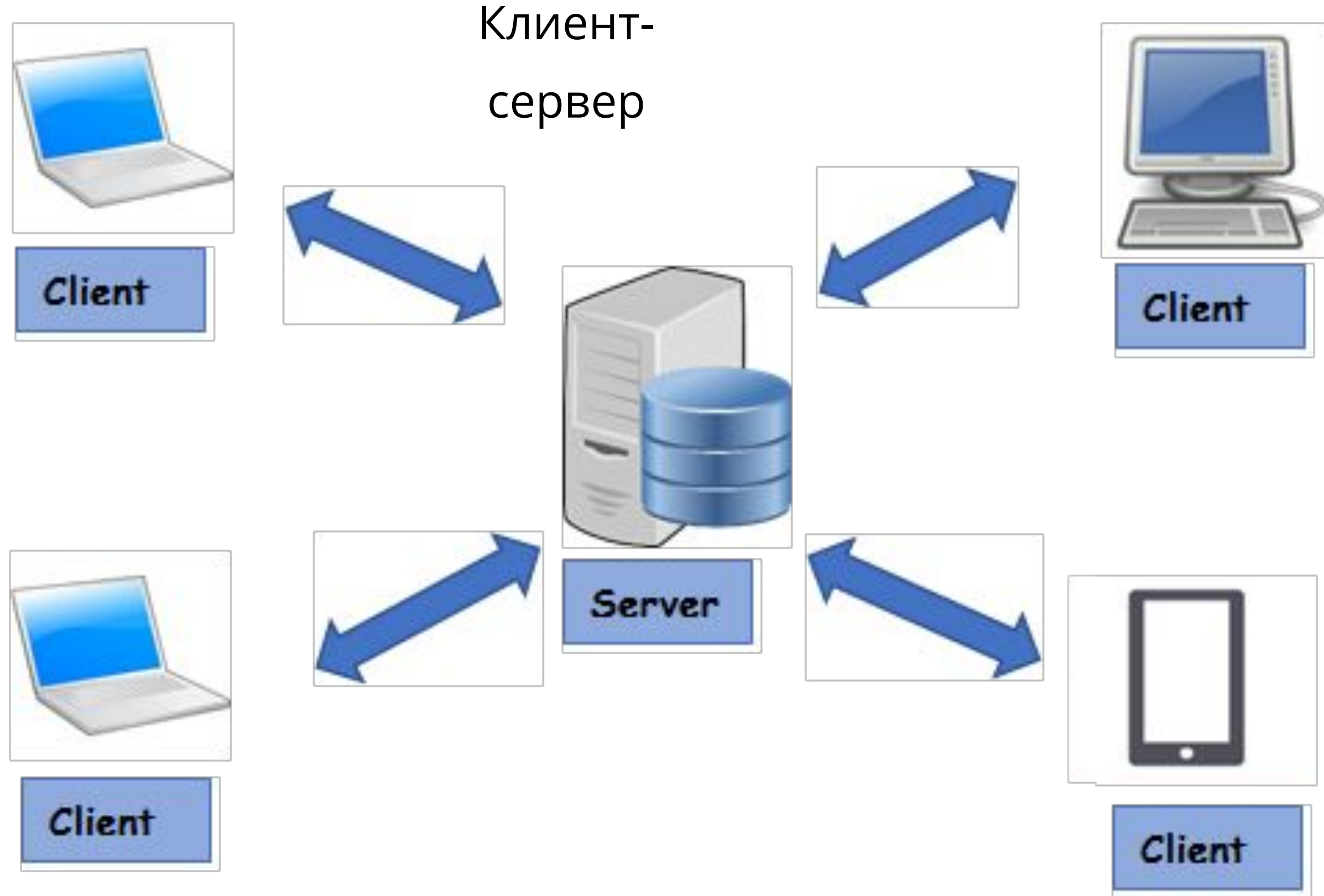

JAVA. СЕТЕВЫЕ ПРИЛОЖЕНИЯ



Сетевое взаимодействие

java.net



Протоколы

https://ru.wikipedia.org/wiki/Таблица_сетевых_протоколов_по_функциональному_назначению

- **Протокол передачи данных** — набор соглашений интерфейса логического уровня, которые определяют обмен данными между различными программами. Эти соглашения задают единообразный способ передачи сообщений и обработки ошибок при взаимодействии программного обеспечения разнесённой в пространстве аппаратуры, соединённой тем или иным интерфейсом.
- Стандартизированный **протокол передачи данных** также позволяет разрабатывать интерфейсы (уже на физическом уровне), не привязанные к конкретной аппаратной платформе и производителю (например, USB, Bluetooth).
- **Сигнальный протокол** используется для управления соединением — например, установки, переадресации, разрыва связи. Примеры протоколов: RTSP, SIP. Для передачи данных используются такие протоколы как RTP.
- **Сетевой протокол** — набор правил и действий (очерёдности действий), позволяющий осуществлять соединение и обмен данными между двумя и более включёнными в сеть устройствами.
- Разные протоколы зачастую описывают лишь разные стороны одного типа связи. Названия «протокол» и «стек протоколов» также указывают на программное обеспечение, которым реализуется протокол.
- Наиболее распространённой системой классификации сетевых протоколов является так называемая **модель OSI**, в соответствии с которой протоколы делятся на 7 уровней по своему назначению
- **Сетевые протоколы** предписывают правила работы компьютерам, подключённым к сети. Они строятся по многоуровневому принципу. Протокол некоторого уровня определяет одно из технических правил связи. В

Модель OSI

- на **физическом уровне** определяются физические (механические, электрические, оптические) характеристики линий связи;
- на **канальном уровне** определяются правила использования физического уровня узлами сети;
- **сетевой уровень** отвечает за адресацию и доставку сообщений;
- **транспортный уровень** контролирует очередность прохождения компонентов сообщения;
- **сеансовый уровень** координирует связь между двумя прикладными программами, работающими на разных рабочих станциях;
- **уровень представления** служит для преобразования данных из внутреннего формата компьютера в формат передачи;
- **прикладной уровень** является пограничным между прикладной программой и другими уровнями, обеспечивая удобный интерфейс связи для сетевых программ пользователя.

Стек протоколов TCP/IP

- канальный уровень (link layer) - Ethernet, DecNet
- сетевой уровень (Internet layer) - IP
- транспортный уровень (transport layer) - TCP, UDP,
- прикладной уровень (application layer) - HTTP, FTP, SMTP, DNS

Наиболее известные протоколы, используемые в сети Интернет:

- HTTP (Hyper Text Transfer Protocol) — это протокол передачи гипертекста. Протокол HTTP используется при пересылке Web-страниц между компьютерами, подключенными к одной сети. 80, 8080
- FTP (File Transfer Protocol) — это протокол передачи файлов со специального файлового сервера на компьютер пользователя. FTP дает возможность абоненту обмениваться двоичными и текстовыми файлами с любым компьютером сети. Установив связь с удаленным компьютером, пользователь может скопировать файл с удаленного компьютера на свой или скопировать файл со своего компьютера на удаленный. 21 100
- POP3 (Post Office Protocol) — это стандартный протокол почтового соединения. Серверы POP обрабатывают входящую почту, а протокол POP предназначен для обработки запросов на получение почты от клиентских почтовых программ. 25
- SMTP (Simple Mail Transfer Protocol) — протокол, который задает набор правил для передачи почты. Сервер SMTP возвращает либо подтверждение о приеме, либо сообщение об ошибке, либо запрашивает дополнительную информацию. 23
- TELNET — это протокол удаленного доступа. TELNET дает возможность абоненту работать на любой ЭВМ находящейся с ним в одной сети, как на своей собственной, то есть запускать программы, менять режим работы и так далее. На практике возможности ограничивается тем уровнем доступа, который задан администратором удаленной

Адрес сервера и сервиса в сети

URL - Uniform Resource Locator - <http://krsu.edu.kg>

Интернет протокол (Internet Protocol - IP) обеспечивает логический адрес, называемый **IP-адресом** сетевого устройства. IP-адрес - это адрес сетевого уровня

Каждому **доменному имени** компьютера (www.ifmo.ru) в **системе доменных имен** (DNS - Domain Name System) соответствует IP-адрес. IP-адреса, используемые в Интернет, имеют определенный формат

Если номер порта не указан, то используется номер порта по умолчанию для сервиса

- TCP обеспечивает надежный канал соединения точка-точка (point-to-point) для клиент/серверных приложений, с помощью которого программы клиента и сервера устанавливают соединение и связывают сокеты.
- Сокеты - это название программного интерфейса хоста для обеспечения информационного обмена между процессами.
- Взаимодействующие процессы могут выполняться как на одном компьютере, так и на различных хостах, соединенных между собой через сеть.
- Сокеты используются для управления каналом связи между приложениями, установленным через сеть.
- Каждое TCP-соединение может быть однозначно идентифицировано своими двумя конечными точками. Таким образом, могут быть обеспечены множественные соединения клиента и сервера. После создания сокета, через него выполняется дальнейшее взаимодействие между клиентом и сервером.

Физический уровень. Ethernet

Кроме IP адреса (или адресов), любое сетевое устройство имеет еще один адрес- адрес физического уровня **MAC-адрес** (от англ. Media Access Control — управление доступом к среде, также Hardware Address, также физический адрес) — уникальный идентификатор, присваиваемый каждой единице активного оборудования или некоторым их интерфейсам в компьютерных сетях Ethernet.

ARP (англ. Address Resolution Protocol — протокол определения адреса) — протокол в компьютерных сетях, предназначенный для определения MAC-адреса по IP-адресу другого компьютера. <https://ru.wikipedia.org/wiki/ARP>

RARP (англ. Reverse Address Resolution Protocol — Обратный протокол преобразования адресов) — протокол сетевого уровня модели OSI, выполняет обратное отображение адресов, то есть преобразует физический адрес в IP-адрес. <https://ru.wikipedia.org/wiki/RARP>

Logical Link Control (общепринятое сокращение — LLC) — подуровень управления логической связью — по стандарту IEEE 802 — верхний подуровень канального уровня модели OSI, осуществляет: управление передачей данных; обеспечивает проверку и правильность передачи информации по соединению. https://ru.wikipedia.org/wiki/Logical_link_control

Другие адреса - URL

Единый указатель ресурса (от англ. **Uniform Resource Locator** — унифицированный указатель ресурса, сокр. **URL**) — система унифицированных адресов электронных ресурсов, или единообразный определитель местонахождения ресурса (файла)[1].

**<схема>:[//[<логин>[:<пароль>]@]<хост>[:<порт>]][/<URL-
путь>][?<параметры>][#<якорь>]**

В этой записи:

- **схема** - схема обращения к ресурсу; в большинстве случаев имеется в виду сетевой протокол логин имя пользователя, используемое для доступа к ресурсу
- **пароль** - пароль указанного пользователя
- **хост** полностью прописанное доменное имя хоста в системе DNS или IP-адрес хоста в форме четырёх групп десятичных чисел, разделённых точками; числа — целые в

URL - пример

protocol://authority@host:port/path/file#ref

protocol://authority@host:port/path/file/extra_path?info

http://www.bhv.ru/

http://132.192.5.10:8080/public/some.html

ftp://guest:password@lenta.ru/users/local/pub

file:///C:/text/html/index.html

- Если какой-то элемент URL отсутствует, то берется стандартное значение.
- Например, в первом примере номер порта port равен 80, а имя файла path — какой-то головной файл, определяемый хостом, чаще всего это файл с именем index.html.
- В третьем примере номер порта равен 21.
- В последнем примере в форме URL просто записано имя файла index.htm, расположенного на разделе C:
Относительный URL жесткого диска той же самой машины.

Другие адреса - URI

URI (англ. **Uniform Resource Identifier**) — унифицированный (единообразный) идентификатор ресурса.

URI — последовательность символов, идентифицирующая абстрактный или физический ресурс.

Ранее назывался Universal Resource Identifier — универсальный идентификатор ресурса.

<https://ru.wikipedia.org/wiki/URI>

URI является либо **URL**, либо **URN**, либо одновременно обоими.

- В URI, как и в URL, можно использовать только ограниченный набор латинских символов и знаков препинания (даже меньший, нежели в ASCII). Если мы захотим использовать в URI символы кириллицы, или иероглифы, или, скажем, специфические символы французского языка, то нам придётся кодировать URI таким же образом, каким в Википедии кодируются URL с символами Юникода.

- Например, строка вида: <https://ru.wikipedia.org/wiki/Кириллица> кодируется в URL как:

<https://ru.wikipedia.org/wiki/%D0%9A%D0%B8%D1%80%D0%B8%D0%BB%D0%BB%D0%B8%D1%86%D0%B0>

Эту проблему призван решить стандарт **IRI** (англ. Internationalized Resource

URI = [схема ":"] ^{Identifier} иерархическая-часть ["?" запрос] ["#" фрагмент

- **схема**: схема обращения к ресурсу (часто указывает на сетевой протокол), например http, ftp, file, ldap, mailto, urn
- **иерархическая-часть**: содержит данные, обычно организованные в иерархической форме, которые, совместно с данными в неиерархическом компоненте запрос, служат для идентификации ресурса в пределах видимости URI-схемы. Обычно иер-часть содержит путь к ресурсу (и, возможно, перед ним, адрес сервера, на котором тот располагается) или идентификатор ресурса (в случае URN).
- **запрос**: этот необязательный компонент URI описан выше.
- **фрагмент**: (тоже необязательный компонент)

Другие адреса - URN

URN (англ. Uniform Resource Name) — единообразное название (имя) ресурса. URN — это постоянная последовательность символов, идентифицирующая абстрактный или физический ресурс. URN является частью концепции URI (англ. Uniform Resource Identifier) — единообразных идентификаторов ресурса. Имена URN призваны в будущем заменить локаторы URL (англ. Uniform Resource Locator) — единообразные определители местонахождения ресурсов. Но имена URN, в отличие от URL, не включают в себя указания на местонахождение и способ обращения к ресурсу. Стандарт URN специально разработан так, чтобы он мог включать в себя другие пространства имён

**<URN> ::= "urn:" <NID> ":"
<NSS>]**

В этой записи:

- <NID> идентификатор пространства имён (англ. Namespace Identifier); представляет собой синтаксическую интерпретацию NSS, не чувствителен к регистру.
- <NSS> строка из определённого пространства имён (англ. Namespace Specific String); если в этой строке содержатся символы не из набора ASCII, то они должны быть закодированы в Юникоде (UTF-8) и предварены (каждый из них) знаком процента «%» (подробнее см. URL).
- При этом начальная последовательность символов "urn: " не чувствительна к регистру. А идентификаторы пространства имён «urn» и «URN» запрещены вообще, чтобы не возникло путаницы с этой начальной строкой "urn: ".

InetAddress

<https://docs.oracle.com/javase/8/docs/api/java/net/InetAddress.html>

IP address это unsigned 32-bit или 128-bit число, которое используется протоколом нижнего уровня IP, на котором построены протоколы UDP и TCP. Экземпляр InetAddress состоит из IP address и иногда имени хоста (если в конструкторе было указано имя хоста или было выполнено обратное распознавание) Типы адресов - unicast и multicast

Доступных пользователю конструкторов нет! Для создания объектов этого класса нужно воспользоваться одним из его **фабричных методов**. В данном случае, у класса InetAddress есть три метода, которые можно использовать для создания представителей. Это методы `getLocalHost`, `getByName` и `getAllByName`.

В приведенном ниже примере выводятся адреса и имена локальной машины, локального почтового узла и WWW-узла

```
InetAddress Address = InetAddress.getLocalHost();  
System.out.println(Address);  
Address = InetAddress.getByName("mailhost");  
System.out.println(Address);  
InetAddress SW[] = InetAddress.getAllByNarne("www.ns.kg");  
System.out.println(SW);
```

У класса InetAddress также есть несколько нестатических методов, которые можно использовать с объектами, возвращаемыми только что названными фабричными методами:

- **getHostName()** возвращает строку, содержащую символическое имя узла, соответствующее хранящемуся в данном объекте адресу Internet.
- **getAddress()** возвращает байтовый массив из четырех элементов, в котором в порядке, используемом в сети, записан адрес Internet, хранящийся в данном объекте.

Package java.net

Класс URL

<https://docs.oracle.com/javase/8/docs/api/java/net/URL.html>

У класса URL из библиотеки Java - четыре конструктора. В наиболее часто используемой форме конструктора URL адрес ресурса задается в строке, идентичной той, которую вы используете при работе с браузером:

```
URL(String spec)    new URL("http://example.com/pages/page1.html");
```

Две следующих разновидности конструкторов позволяют задать URL, указав его отдельные компоненты:

```
URL(String protocol, String host, int port, String file)
```

```
URL gamelan = new URL("http", "example.com", 80, "pages/page1.html")
```

```
URL(String protocol, String host, String file)    new URL("http", "example.com", "/pages/page1.html")
```

Четвертая, и последняя форма конструктора позволяет использовать существующий URL в качестве ссылочного контекста, и создать на основе этого контекста новый URL.

```
URL(URL context, String spec) или URL(URL baseURL, String relativeURL)
```

```
URL myURL = new URL("http://example.com/pages/");
```

```
URL page1URL = new URL(myURL, "page1.html");
```

```
URL page2URL = new URL(myURL, "page2.html");
```

```
URL page1BottomURL = new URL(page1URL, "#BOTTOM");
```


Package `java.net`

Класс `URL`

<https://docs.oracle.com/javase/8/docs/api/java/net/URL.html>

`MalformedURLException`

Each of the four `URL` constructors throws a `MalformedURLException` if the arguments to the constructor refer to a null or unknown protocol.

Typically, you want to catch and handle this exception by embedding your `URL` constructor statements in a try/catch pair, like this:

```
try {
    URL myURL = new URL(...);
}
catch (MalformedURLException e) {
    // exception handler code here
    // ...
}
```

Пример

- В приведенном ниже примере создается URL, адресующий www-страницу (поставьте туда свой адрес), после чего программа печатает свойства этого объекта.

```
import java.net.URL;

class myURL {
    public static void main(String args[]) throws Exception {
        URL hp = new URL("http://coop.chuvashia.edu");
        System.out.println("Protocol: " + hp.getProtocol());
        System.out.println("Port: " + hp.getPort());
        System.out.println("Host: " + hp.getHost());
        System.out.println("File: " + hp.getFile());
        System.out.println("Ext: " + hp.toExternalForm());
    }
}
```

Прямое чтение

After you've successfully created a URL, you can call the URL's **openStream()** method to get a stream from which you can read the contents of the URL. The `openStream()` method returns a `java.io.InputStream` object, so reading from a URL is as easy as reading from an input stream.

The following small Java program uses `openStream()` to get an input stream on the URL `http://www.oracle.com/`. It then opens a `BufferedReader` on the input stream and reads from the `BufferedReader` thereby reading from the URL. Everything read is copied to the standard output stream:

```
import java.net.*;
import java.io.*;
public class URLReader {
    public static void main(String[] args) throws Exception {
        URL oracle = new URL("http://www.oracle.com/");
        BufferedReader in = new BufferedReader(
            new InputStreamReader(oracle.openStream()));
        String inputLine;
        while ((inputLine = in.readLine()) != null)
            System.out.println(inputLine);
        in.close();
    }
}
```

When you run the program, you should see, scrolling by in your command window, the HTML commands and textual content from the HTML file located at `http://www.oracle.com/`. Alternatively, the program might hang or you might see an exception stack trace. If either of the latter two events

URLConnection

<https://docs.oracle.com/javase/8/docs/api/java/net/URLConnection.htm>

Для того, чтобы извлечь реальную информацию, адресуемую данным URL, необходимо на основе URL создать объект `URLConnection`, воспользовавшись для этого методом `openConnection()`.

`URLConnection` — объект, который используется либо для проверки свойств удаленного ресурса, адресуемого URL, либо для получения его содержимого.

В приведенном ниже примере **`URLConnection`** создается с помощью метода **`openConnection`**, вызванного с объектом URL. После этого созданный объект используется для получения содержимого и свойств документа.

Эта программа устанавливает HTTP-соединение с локальным узлом по порту 80 (на машине должен быть установлен Web-сервер) и запрашивает документ по умолчанию, обычно это - `index.html`. После этого программа выводит значения заголовка, запрашивает и выводит содержимое документа.

Sockets

Сокет - точка соединения двусторонней линии связи между процессами в сети.

Datagram Sockets

stateless без установки

User Datagram Protocol

Датаграмма - пакет данных, отправленный по сети, прибытие которого, время прибытия и содержание не гарантировано. Не гарантируется также и порядок доставки пакетов. При передаче пакета UDP по какому-либо адресу нет никакой гарантии того, что он будет принят, а также, что по этому адресу вообще существует потребитель пакетов.

Аналогично, когда вы получаете датаграмму, у вас нет никаких гарантий, что она не была повреждена в пути следования или что отправитель ожидает подтверждения получения датаграммы. Использование UDP может привести к потере или к дублированию пакетов, что приводит к дополнительным проблемам, связанным с проверкой ошибок и обеспечением надежности передачи данных. Если вам необходимо добиться оптимальной производительности, и вы готовы сократить затраты на проверку целостности информации, пакеты UDP могут оказаться весьма полезными

Stream Sockets

statefull с установкой

Transmission Control

Protocol/Internet Protocol

соединения

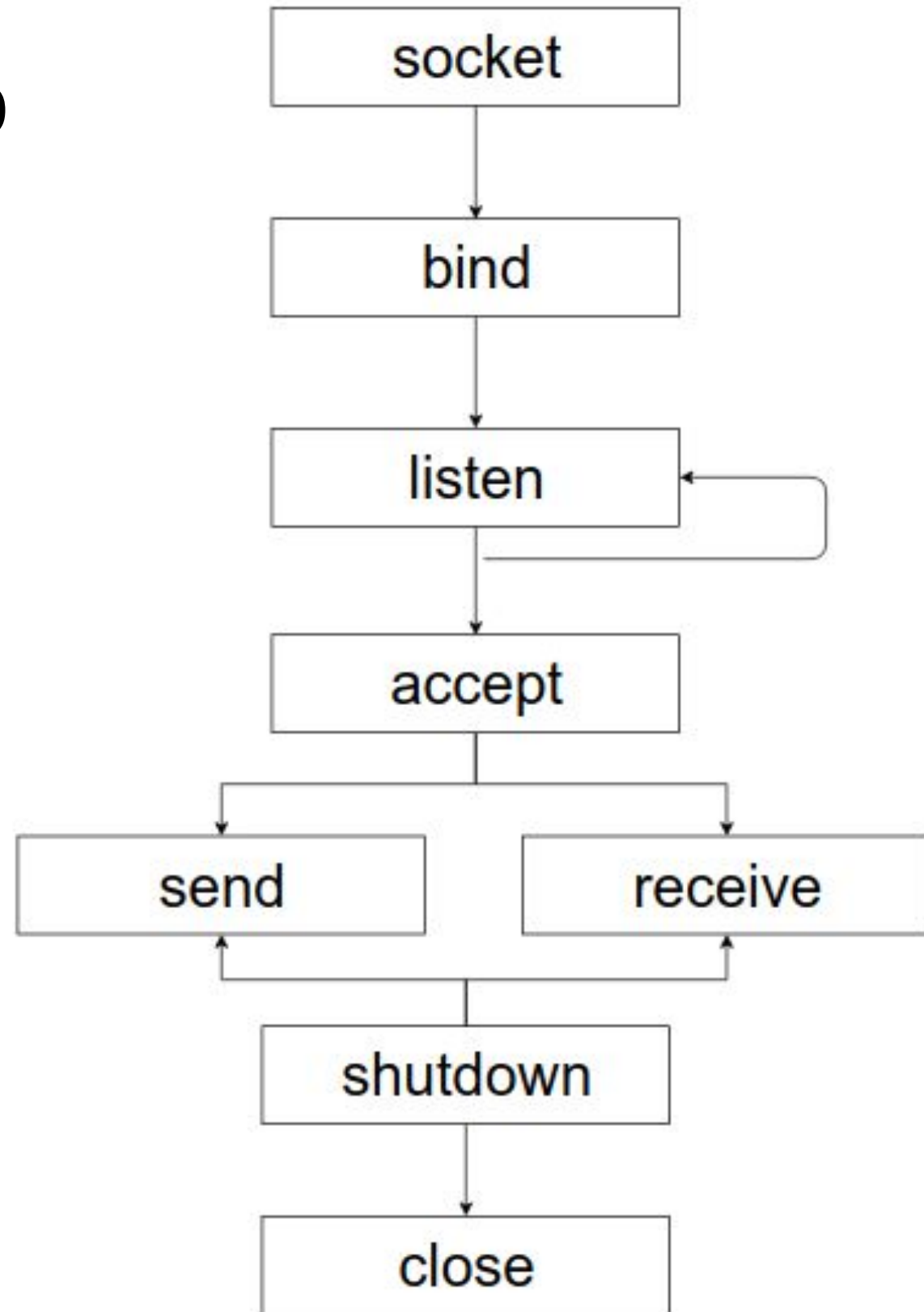
При использовании потоковых сокетов, программа устанавливает соединение с другим сокетом и, пока соединение установлено, поток данных протекает между программами, и говорят, что потоковые сокет обеспечивают обслуживание на основе установления соединения.

TCP/IP является потоковым протоколом на основе установления двунаправленных соединений точка-точка между узлами Интернет, и взаимодействие между компьютерами по этому протоколу предназначено для реализации надежной передачи данных. Все данные, отправленные по каналу передачи, получаются в том же порядке, в котором они передавались. В отличие от датаграммных сокетов, сокеты TCP/IP реализуют высоконадежные устойчивые соединения между клиентом и сервером.

Сокеты с установкой соединения

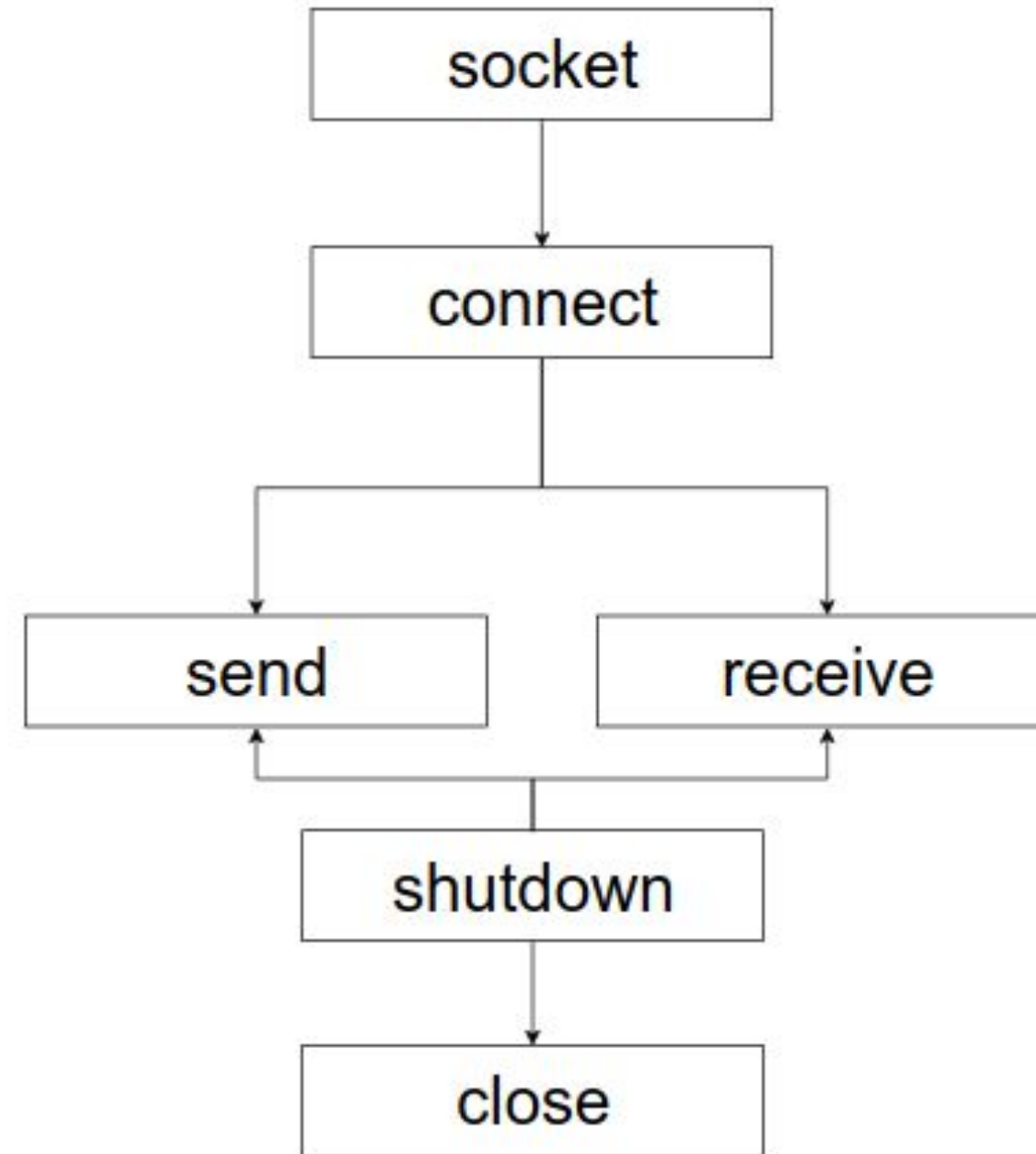
Сервер

р

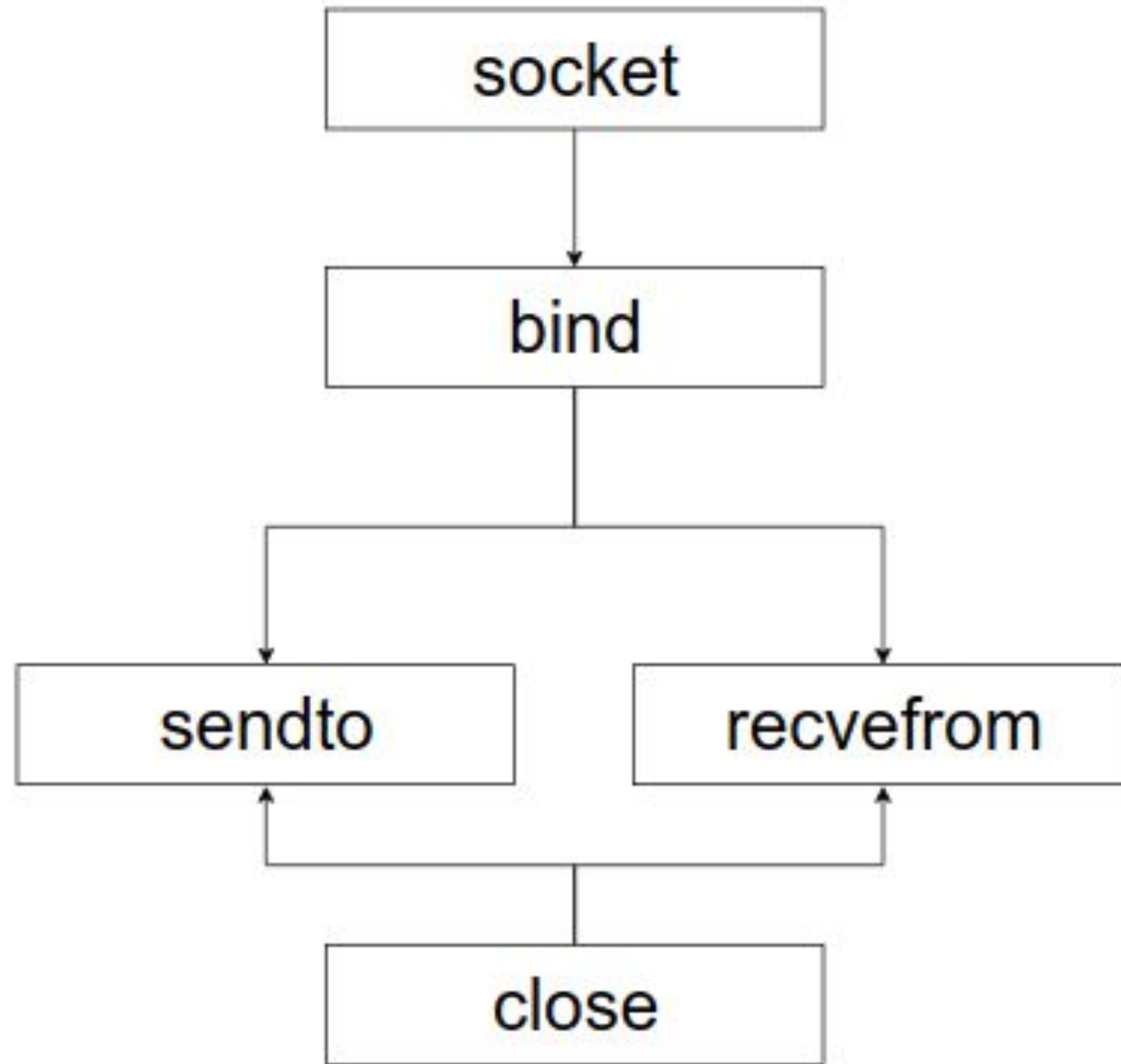


Клиент

т



Сокеты без установки соединения



Package java.net

<https://docs.oracle.com/javase/8/docs/api/java/net/package-summary.html>

Class Socket

<https://docs.oracle.com/javase/8/docs/api/java/net/Socket.html>

Class ServerSocket

<https://docs.oracle.com/javase/8/docs/api/java/net/ServerSocket.htm>

|

- ServerSocket ждет, пока клиент не установит с ним соединение, в то время, как обычный Socket трактует недоступность чего-либо, с чем он хочет соединиться, как ошибку.
- Одновременно с созданием объекта Socket устанавливается соединение между узлами Internet.

"Серверные" сокеты

Для создания серверов Internet надо использовать объекты класса **ServerSocket**. Когда вы создаете объект `ServerSocket`, он регистрирует себя в системе, говоря о том, что он готов обслуживать соединения клиентов.

У этого класса есть один дополнительный метод **accept()**, вызов которого блокирует подпроцесс до тех пор, пока какой-нибудь клиент не установит соединение по соответствующему порту. После того, как соединение установлено, метод `accept()` возвращает вызвавшему его подпроцессу обычный объект `Socket`.

Два конструктора класса `ServerSocket` позволяют задать, по какому порту вы хотите соединяться с клиентами, и (необязательный параметр) как долго вы готовы ждать, пока этот порт не освободится.

- **ServerSocket(int port)** создает сокет сервера для заданного порта.
- **ServerSocket(int port, int count)** создает сокет сервера для заданного порта. Если этот порт занят, метод будет ждать его освобождения максимум `count` миллисекунд.

"Серверные" сокеты - конструкторы

ServerSocket()

Creates an unbound server socket.

ServerSocket(int port)

Creates a server socket, bound to the specified port.

ServerSocket(int port, int backlog)

Creates a server socket and binds it to the specified local port number, with the specified backlog.

ServerSocket(int port, int backlog, InetAddress bindAddr)

Create a server with the specified port, listen backlog, and local IP address to bind to.

"Серверные" сокеты - метод accept

<https://docs.oracle.com/javase/8/docs/api/java/net/ServerSocket.html#accept->

public Socket accept()

throws IOException

Listens for a connection to be made to this socket and accepts it. The method blocks until a connection is made.

A new Socket *s* is created and, if there is a security manager, the security manager's `checkAccept` method is called with `s.getInetAddress().getHostAddress()` and `s.getPort()` as its arguments to ensure the operation is allowed. This could result in a `SecurityException`.

Returns: the new Socket

Throws: `IOException` - if an I/O error occurs when waiting for a connection.

`SecurityException` - if a security manager exists and its `checkAccept` method doesn't allow the operation.

`SocketTimeoutException` - if a timeout was previously set with `setSoTimeout` and the timeout has been reached.

`IllegalBlockingModeException` - if this socket has an associated channel, the channel is in

"Клиентские"

Для создания сокетов вы можете использовать два конструктора:

СОКЕТЫ

- **Socket(String host, int port)** устанавливает соединение между локальной машиной и указанным портом узла Internet, имя которого было передано конструктору. Этот конструктор может возбуждать исключения UnknownHostException и IOException.
- **Socket(InetAddress address, int port)** выполняет ту же работу, что и первый конструктор, но узел, с которым требуется установить соединение, задается не строкой, а объектом InetAddress. Этот конструктор может возбуждать только IOException.

Из объекта Socket в любое время можно извлечь информацию об адресе Internet и номере порта, с которым он соединен. Для этого служат следующие методы:

- **getInetAddress()** возвращает объект InetAddress, связанный с данным объектом Socket.
- **getPort()** возвращает номер порта на удаленном узле, с которым установлено соединение.
- **getLocalPort()** возвращает номер локального порта, к которому присоединен данный объект.

После того, как объект Socket создан, им можно воспользоваться для того, чтобы получить доступ к связанным с ним входному и выходному **потокам**. Эти потоки используются для приема и передачи данных точно так же, как и обычные потоки ввода-вывода:

- **getInputStream()** возвращает InputStream, связанный с данным объектом.
- **getOutputStream()** возвращает OutputStream, связанный с данным объектом.
- **close()** закрывает входной и выходной потоки объекта Socket.

Пример

Приведенный ниже очень простой пример открывает соединение с портом 880 сервера "timehost" и выводит полученные от него данные.

```
import java.net.*;
import java.io.*;
class TimeHost {
public static void main(String args[]) throws Exception {
int c;
Socket s = new Socket("timehost.starwave.com",880);
InputStream in = s.getInputStream();
while ((c = in.read()) != -1) {
System.out.print( (char) c);
}
s.close();
} }
```

Многопоточный сервер

МНОГОПОТОЧНЫЙ сервер

```
import java.io.IOException;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket;
public class MultiThreadServer implements Runnable {
    Socket csocket;
    MultiThreadServer(Socket csocket) {
        this.csocket = csocket;
    }
    public static void main(String args[]) throws Exception {
        ServerSocket ssock = new ServerSocket(1234);
        System.out.println("Listening");

        while (true) {
            Socket sock = ssock.accept();
            System.out.println("Connected");
            new Thread(new MultiThreadServer(sock)).start();
        }
    }
}
```

```
public void run() {
    try {
        PrintStream pstream = new PrintStream(csocket.getOutputStream());
        for (int i = 100; i >= 0; i--) {
            pstream.println(i + " bottles of beer on the wall");
        }
        pstream.close();
        csocket.close();
    } catch (IOException e) {
        System.out.println(e);
    }
}
}
```

<https://docs.oracle.com/javase/tutorial/networking/sockets/index.html>

<http://www.realcoding.net/articles/glava-19-setevye-sredstva-java.html>

<https://docs.oracle.com/javase/tutorial/networking/sockets/clientServer.html>

<http://shtanyuk.tk/edu/nniit/java-new/html/11.html>