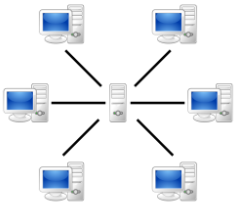
# Основи на мрежовото програмиране

Николай Ланджев / SAP Labs Bulgaria
Ноември 2013

Public

# Съдържание

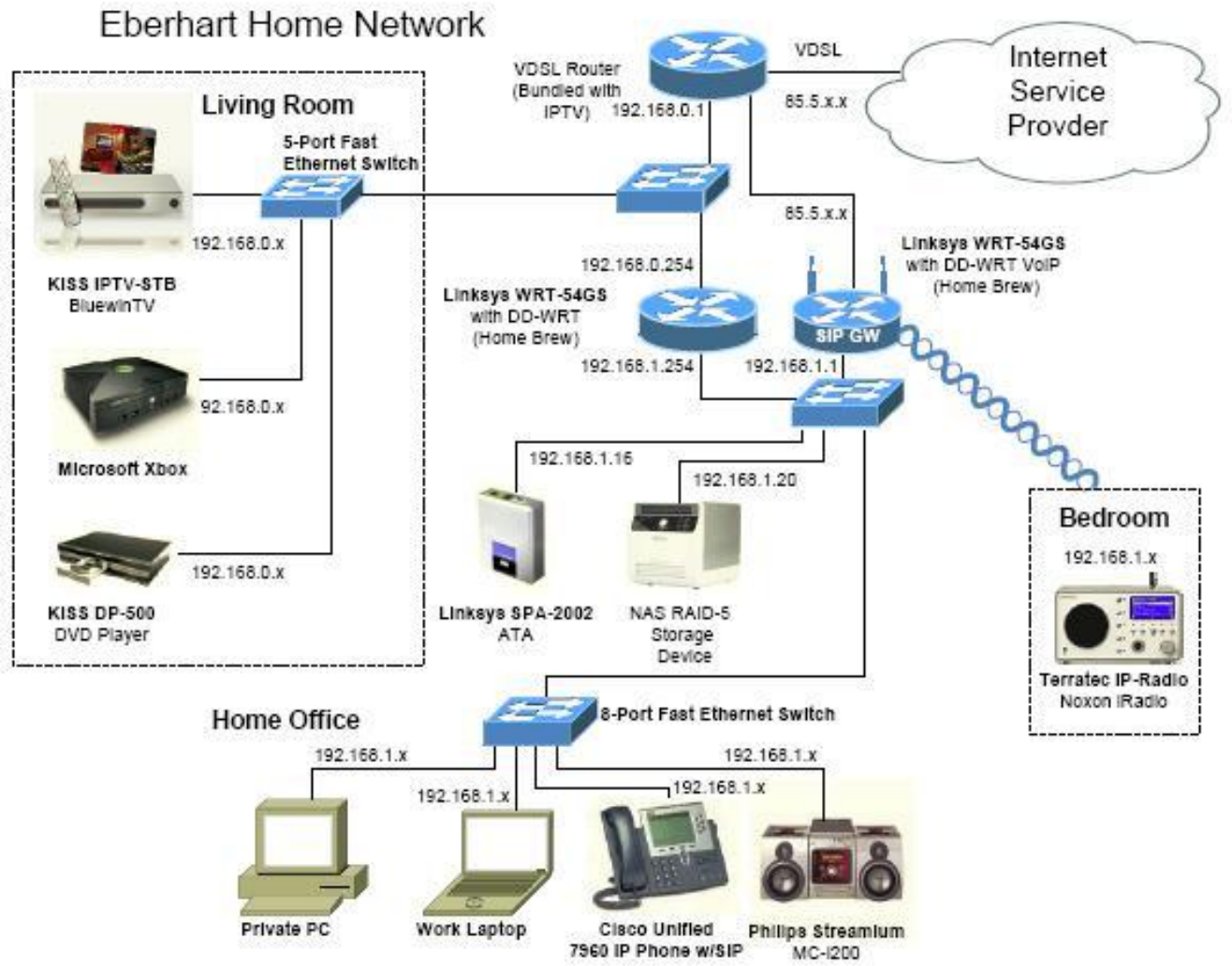
Мрежови основи



Моделът клиент-сървър

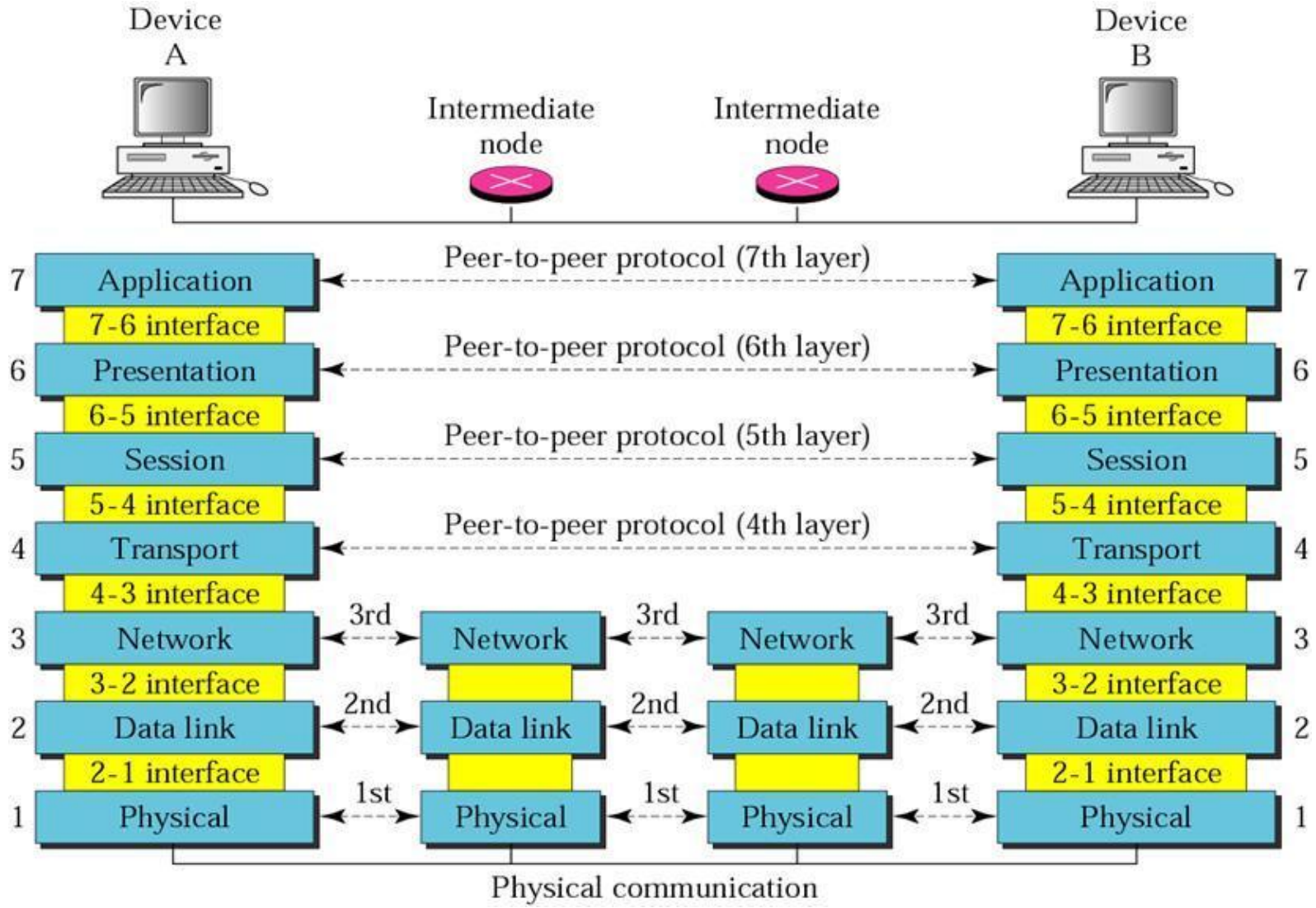

Мрежова комуникация с Java

# Мрежови основи
## Мрежата



# Мрежови основи

## Предаване на данни



# Мрежови основи

## OSI (Open Systems Interconnection) модел

| № | Слой | Описание | Протоколи |
|---|---|---|---|
| 7 | Application | Позволява на потребителските приложения да заявяват услуги или информация, а на сървър приложенията — да се регистрират и предоставят услуги в мрежата. | DNS, FTP, HTTP, NFS, NTP, DHCP, SMTP, Telnet |
| 6 | Presentation | Конвертиране, компресиране и криптиране на данни. | TLS/SSL |
| 5 | Session | Създаването, поддържането и терминирането на сесии. Сигурност. Логически портове. | Sockets |
| 4 | Transport | Грижи се за целостта на съобщенията, за пристигането им в точна последователност, потвърждаване за пристигане, проверка за загуби и дублиращи се съобщения. | TCP, UDP |
| 3 | Network | Управлява на пакетите в мрежата. Рутиране. Фрагментация на данните. Логически адреси. | IPv4, IPv6, IPX, ICMP |
| 2 | Data Link | Предаване на фреймове от един възел на друг. Управление на последователността на фреймовете. Потвърждения. Проверка за грешки. MAC. | ATM, X.25, DSL, IEEE 802.11 |
| 1 | Physical | Отговаря за предаването и приемането на неструктурирани потоци от данни по физическият носител. Кодиране/декодиране на данните. Свързване на физическият носител. | IEEE 802.11, IEEE 1394, Bluetooth |

# Мрежови основи
## TCP/UDP

| Transmission Control Protocol (TCP) | User Datagram Protocol (UDP) |
|---|---|
| **Надеждност:** TCP е протокол, който се основава на връзки (connections). Когато един файл се изпрати, той със сигурност ще бъде доставен, освен ако връзката не се прекрати. Ако част от пакетите бъдат изгубени, те ще бъдат предадени отново. | **Надеждност:** UDP е пакетно ориентиран протокол. Когато се изпрати съобщение по мрежата, не е сигурно дали то ще бъде доставено или дали ще запази своята цялост. |
| **Подредба:** Ако се изпратят две последователни съобщения, протокола гарантира, че те ще се получат в реда в който са изпратени. | **Подредба:** Протокола не гарантира, че съобщенията ще се получат в реда в който са изпратени. |
| **Тежък:** Протокола изисква допълнителни мрежови трафик, за потвърждения и изпращане отново на пакети, които са се загубили по мрежата или които не са доставени в правилният ред. | **Лек:** Няма подредба на съобщенията или потвърждаване за получените пакети. |
| **Streaming:** Данните се четата като "stream,“. Може да се изпратят/получат няколко пакета едновременно. | **Datagrams:** Пакетите се изпращат индивидуално. |
| **Примери:** HTTP, SMTP, FTP, SSH и други. | **Примери:** DNS, IPTV, VoIP, TFTP и други. |

# Мрежови основи

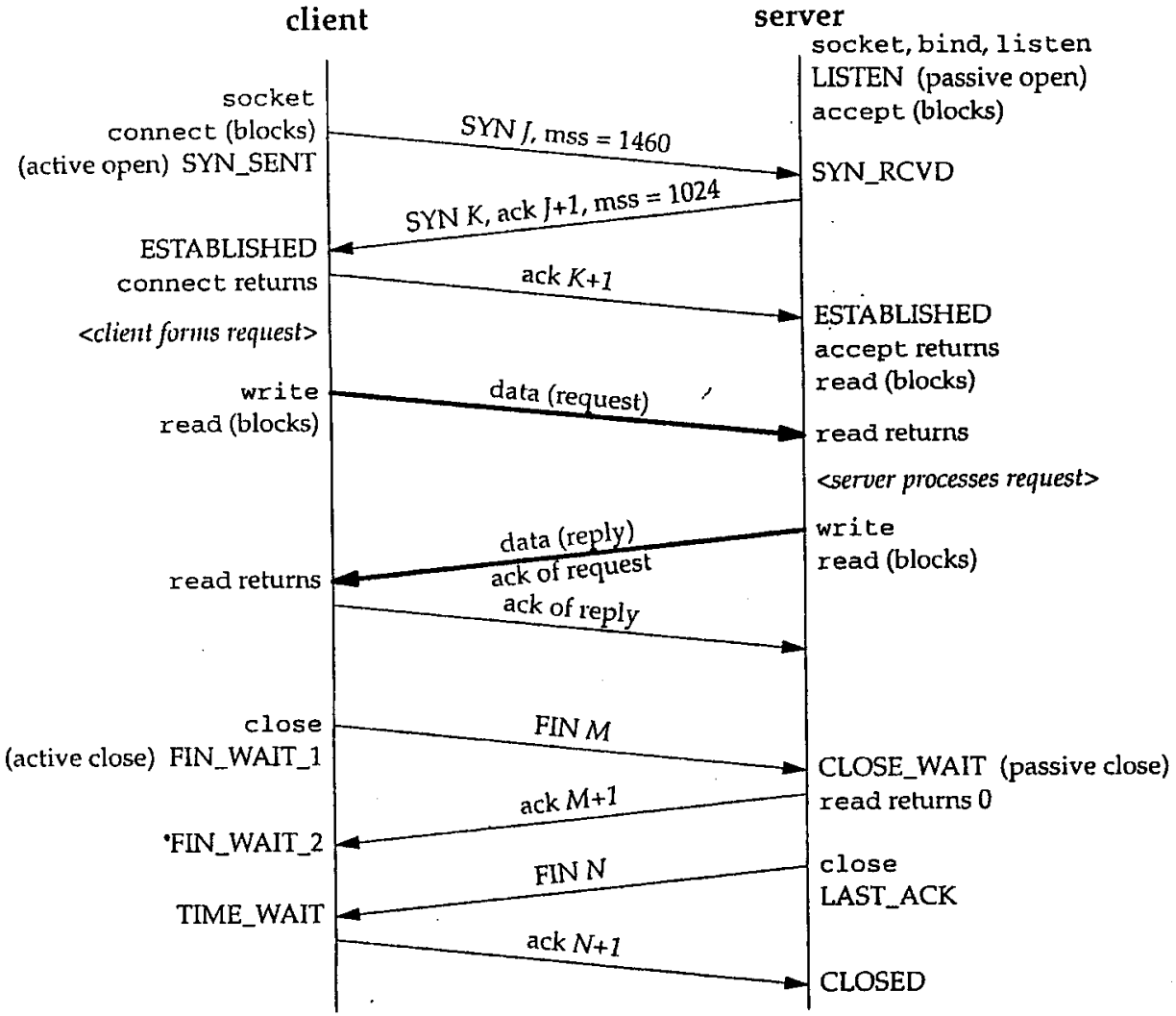
## Кога да използваме UDP

Поради характеристиките на UDP протокола, е добре да се използва при:

- **Broadcast или multicast приложения.** В общият случай, това са приложения, които се опитват да открият някакъв ресурс в мрежата или да проверят кои клиенти са активни.
- **При предаване на данни, които ще бъдат заменени скоро** с нови данни: Windows 8 tiles.
- **При предаване на данни, които не са критични**. Например при streaming на видео или VoIP (Skype).
- **Приложения които ще обработват огромно количество заявки** (request), които не създават сесия. DNS
- Не трябва да се използва за трансфер на голямо количество данни.

# Мрежови основи

## Как работи TCP протокола



# Мрежови основи

## Идентифициране на приложение

Как идентифицираме един компютър в мрежата?

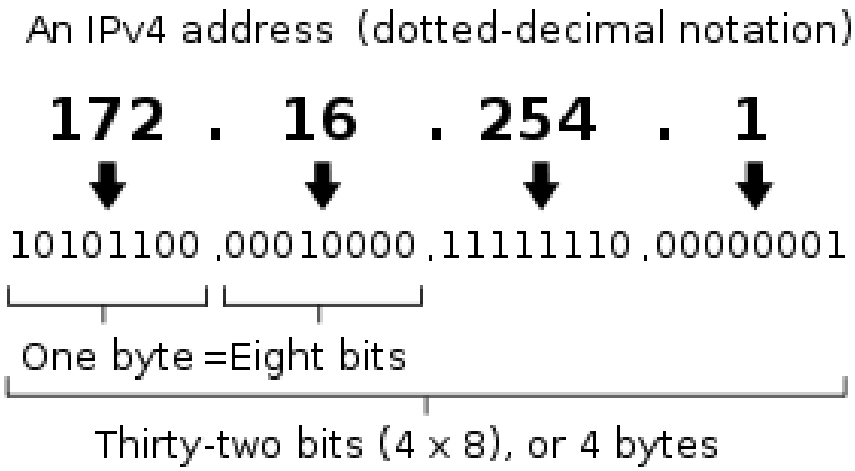

| 10.199.199.200 | 50430 |
|---|---|
| IP Адрес | Порт |

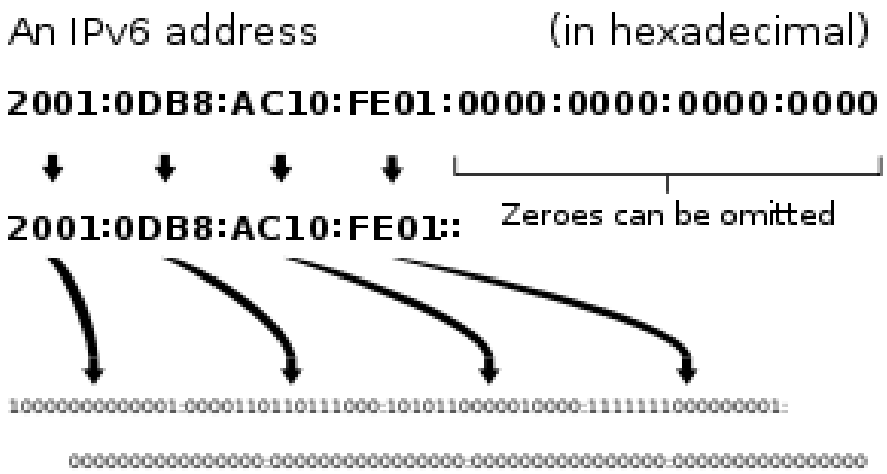
Socket

# Мрежови основи

## IP протокол

- Всеки компютър свързан към която и да е мрежа се идентифицира със логически адрес.
- Най-разпространените протоколи за логически адреси в мрежата са IP (Internet Protocol) версия 4 и IP версия 6.
- Адресите в IPv4 представляват 32 битови числа, а в IP v6 128 битови.



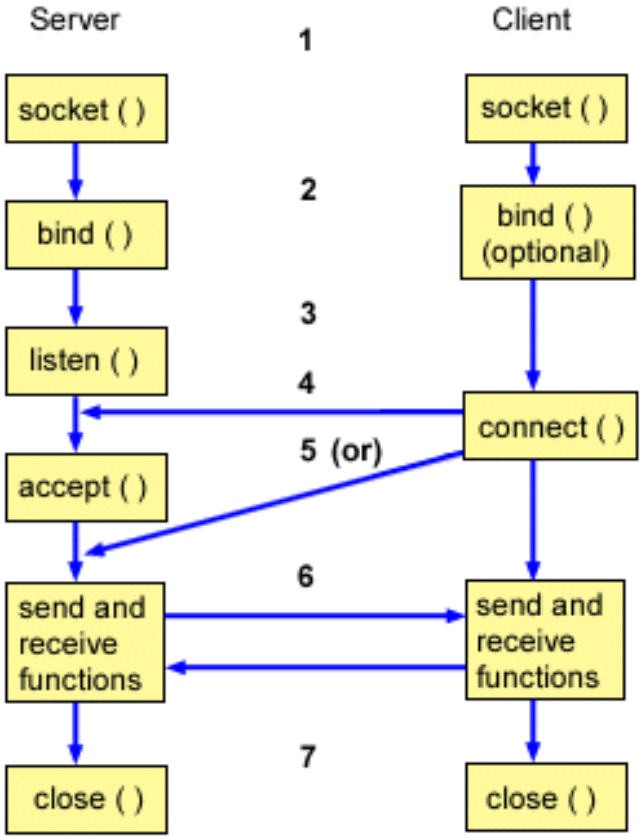
# Мрежови основи
## Портове

- В общият случай, компютърът има една физическа връзка към мрежата. По тази връзка се изпращат и получават данни от/за всички приложения. Портовете се използват, за да се знае кои данни за кое приложение са.
- Предадените данни по мрежата, винаги съдържат в себе си информация за компютъра и порта към които са насочени.
- Портовете се идентифицират с 16 битово число, което се използва от UDP и TCP протокола, да идентифицират за кое приложение се предназначени данните.
- Портовете могат да бъдат от номер 0 до номер 65 535.
- Портове с номера от 0 до 1023 са известни като “well-known ports”. За да се използват тези портове от вашето приложение, то трябва да се изпълнява с администраторски права.

# Мрежови основи
## Сокети (Socket)

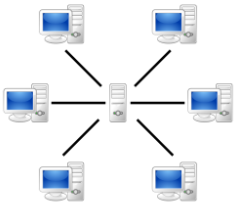
Сокетите се използват при клиент-сървър комуникация.
Сокета представлява една крайна точка от двупосочна връзка.
Сокет = хост + порт
Състояния на сокетите:

1. socket() метода създава крайна точка за комуникация и връща дескриптор на сокета.
2. bind() метода създава уникално име за този сокет. По този начин сървъра е достъпен по мрежата.
3. listen() метода показва готовност за приемане на клиентски връзки.
4. Клиентското приложение трябва да извика метода connect(), за да осъществи връзка със сървъра.
5. Сървърно приложение използва accept() метода, за да приеме връзка от клиента.
6. След като е осъществена връзката, може да се използват методите за трансфер на потоци (streams): send(), recv(), read(), write() и други.
7. Сървъра или клиента може да прекратят връзката с метода close().



# Съдържание

Мрежови основи



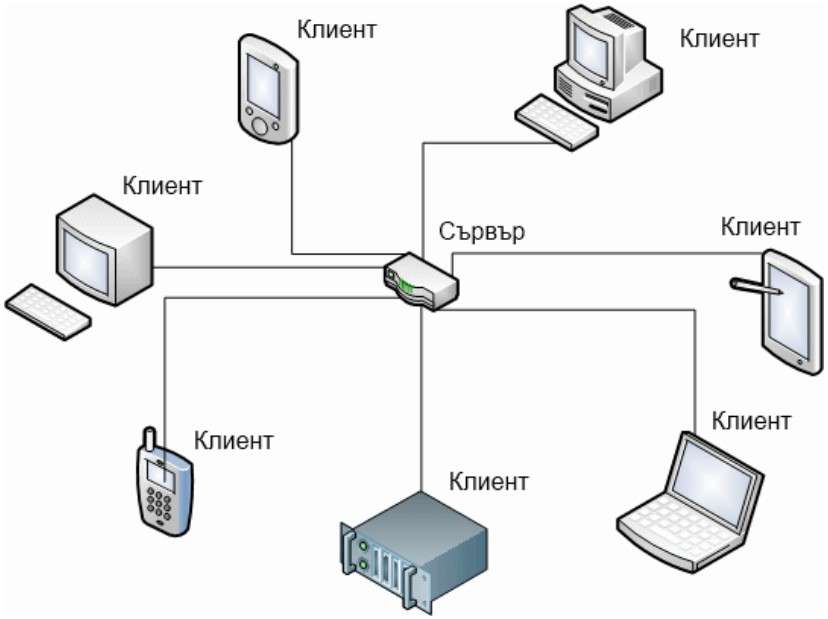
Моделът клиент-сървър

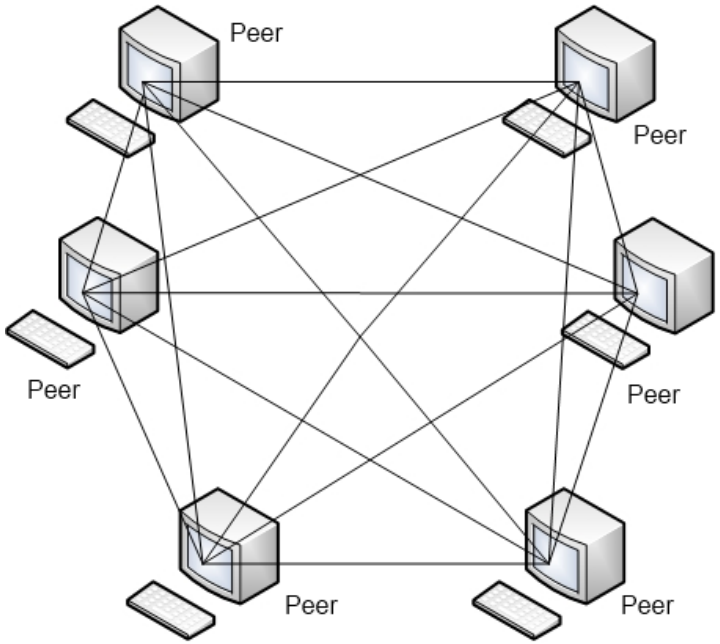

Мрежова комуникация с Java

# Моделът клиент-сървър

## Архитектурни модели

- **Клиент-сървър** е разпределен изчислителен модел, при който част от задачите се разпределят между доставчиците на ресурси или услуги, наречени ***сървъри*** и консуматорите на услуги, наречени ***клиенти***.




- **Peer-to-peer** е разпределен архитектурен модел на приложение, при който задачите се разпределят по еднакъв начин между всички участници (peer, node). Всеки участник е едновременно и **клиент** и **сървър**.

# Моделът клиент-сървър

## Клиент-сървър – Клиенти

Според наличната функционалност в клиента:

- **Rich** клиенти.
- **Thin** клиенти.

Според семантиката (протокол):

- **Web** клиенти – Браузери (Chrome, Firefox, IE).
- **Mail** клиенти – POP/SMTP клиенти (MS Outlook, Lotus notes).
- **FTP** клиенти – Total Commander, Filezilla, WinSCP.
- …

# Моделът клиент-сървър

## Клиент-сървър - Сървъри

**Файл** сървър (Windows, Samba, UNIX NFS, OpenAFS).

**DB** сървър (MySQL, PostgreSQL, Oracle, MS SQL Server, Mongo DB, HANA).

**Mail** сървър (MS Exchange, GMail, Lotus Notes).

**Name** сървър (DNS).

**FTP** сървър (ftpd, IIS).

**Print** сървър.

**Game** сървър.

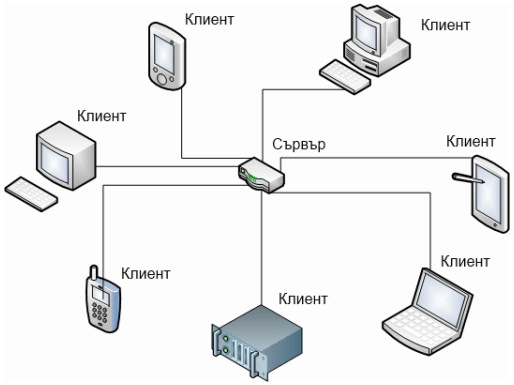
**Web** сървър (Apache, GWS, MS IIS, nginx).

**Application** сървър (SAP NetWeaver, Tomcat, GlassFish, JBoss, BEA, Oracle).

…

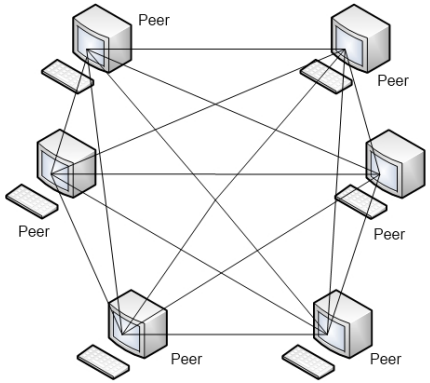
# Моделът клиент-сървър

## Предимства и недостатъци




### Клиент-сървър

- Single Point Of Failure (SPOF).
- Увеличаването на броя на клиентите води до намаляване на производителността.
- 70-95% от времето, през което работи сървъра е idle.




### Peer-to-peer

+ Няма SPOF.
+ Няма намаляване на производителността при увеличаване на клиентите.
- Проблеми със сигурността.
- Риск от умишлена промяна на съдържание.
- Липса на контрол върху съдържанието и възможност за загуба на съдържание.
- Труден процес на поддръжка.

# Моделът клиент-сървър

## Клиент-сървър – Еволюция на модела

### Клъстъри

- Няколко компютъра свързани помежду си, които в много отношения могат да се разглеждат като една система (сървър).
- Всеки елемент (компютър) в клъстера има собствена операционна система и сървър софтуер.
- Отделните елементи обикновено са свързани помежду си с локална мрежа.
- Обикновено се използват за решаване на известни проблеми на клиент-сървър модела: подобряване на производителността и капацитета, по-висока надеждност.

### GRID

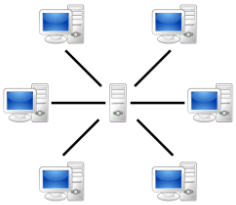
### Supercomputers

### Cloud

- Гъвкав бизнес модел.
- Подобряване на ефективността чрез виртуализация.

# Съдържание

Мрежови основи



Моделът клиент-сървър


Мрежова комуникация с Java

# Мрежова комуникация с Java
## Въведение

Пакета **java.net** предоставя класове, които използват и работят на различни нива от OSI модела.

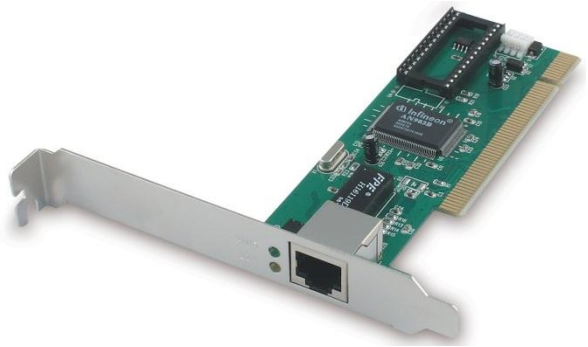
- **Мрежов** и **data link** слой – класа **NetworkInterface** предоставя достъп до мрежовите адаптери на компютъра.
- **Транспортен** слой – В зависимост от използваните класове може да се използват следните транспортни протоколи:
  - **TCP** – класове **Socket** и **ServerSocket**.
  - **UDP** – класове **DatagramPacket**, **DatagramSocket**, **MulticastSocket**.
- **Приложен** слой – класовете **URL** и **URLConnection** се използват за достъпването на **HTTP** и **FTP** ресурси.

Класовете в пакета **java.io** се използват за обработка на потоци от данни (**data streams**):

**InputStream**, **BufferedInputStream**, **Reader**, **InputStreamReader**, **BufferedReader**, **OutputStream**, **BufferedOutputStream**, **Writer**, **PrintWriter** са класове, които често се използват при мрежовото програмиране в Java.

# Мрежова комуникация с Java
## Мрежови адаптери (1)

- Мрежовият адаптер осъществява връзката между компютърната система и публична или частна мрежа.
- Мрежовите адаптери могат да бъдат физически или виртуални (софтуерни). Примери за виртуални са loopback интерфейса и интерфейсите създадени от виртуалните машини.
- Една система може да има повече от един физически и/или виртуален мрежови адаптер.
- Java предоставя достъп до всички мрежови адаптери чрез класа **java.net.NetworkInterface**.



# Мрежова комуникация с Java

## Мрежови адаптери (2)

С помощта на класа NetworkInterface може да вземете списък със всички мрежови адаптери (getNetworkInterfaces()) или да вземете точно определен (getByInetAddress() и getByName()).

```
Enumeration<NetworkInterface> nets = NetworkInterface.getNetworkInterfaces();
for (NetworkInterface netIf : Collections.list(nets)) {
   System.out.printf("Display name: %s\n", netIf.getDisplayName());
   System.out.printf("Name: %s\n", netIf.getName());
   System.out.printf("Addresses: %s\n", printEnum(netIf.getInetAddresses()));
   System.out.printf("\n");
}
```

*Display name: Software Loopback Interface 1*
*Name: lo*
*Addresses: /0:0:0:0:0:0:0:1, /127.0.0.1,*

*Display name: Intel(R) 82578DM Gigabit Network Connection*
*Name: eth6*
*Addresses: /10.xxx.xxx.xxx,*

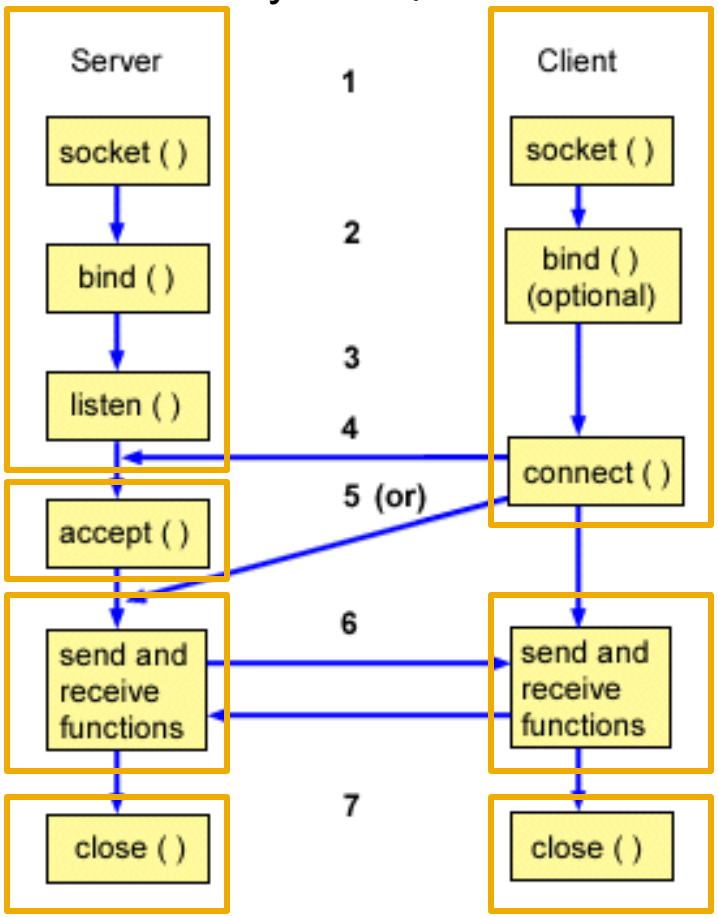
*Display name: WAN Miniport (IPv6)-QoS Packet Scheduler-0000*
*Name: eth14*
*Addresses:*

# Мрежова комуникация с Java

## TCP комуникация

Използват се два класа за TCP комуникацията: Socket и ServerSocket

```
ServerSocket serSock =
new ServerSocket(4444);

Socket sock =
serverSocket.accept();

OutputStream out =
sock.getOutputStream();
InputStream in =
sock.getInputStream();

out.close();
in.close();
sock.close();
serSock.close();
```



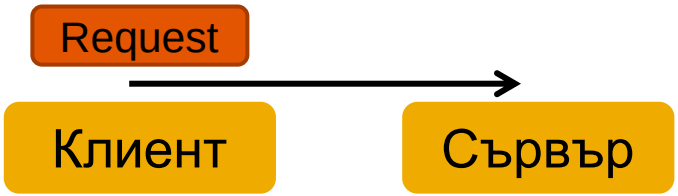

```
Socket sock = new
Socket(“hostname”, 4444);

OutputStream out =
sock.getOutputStream();
InputStream in =
sock.getInputStream();

out.close();
in.close();
sock.close();
```

# Мрежова комуникация с Java

## Клиент-сървър комуникация (1)

Отваряне на сокет от клиентската страна и изпращане на заявка (**request**):

```
try {
    Socket s = new Socket("www.tu-sofia.bg", 80);
    PrintWriter pw = new PrintWriter(s.getOutputStream());
    pw.println("GET /index.html");
    pw.println();
    pw.flush();
} catch (UnknownHostException e) {
} catch (IOException e) {
}
```

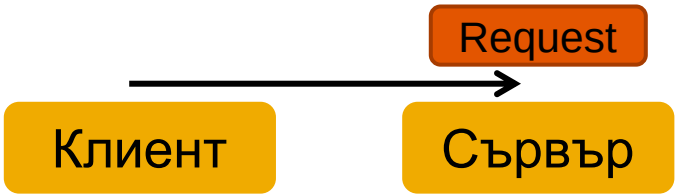


# Мрежова комуникация с Java

## Клиент-сървър комуникация (2)

Отваряне на сокет от страна на сървъра и приемане на заявки :

```
try {
    ServerSocket ss = new ServerSocket(80);
    Socket s = ss.accept(); //The thread is blocked.
    //New connection is established. Read the request
    BufferedInputStream is = new BufferedInputStream(s.getInputStream());
    ByteArrayOutputStream byteOs = new ByteArrayOutputStream();
    byte[] b = new byte[2048];
    int r = 0;
    while ((r = is.read(b)) != -1) {
        byteOs.write(b, 0, r);
    }
} catch (IOException e) {
}
```

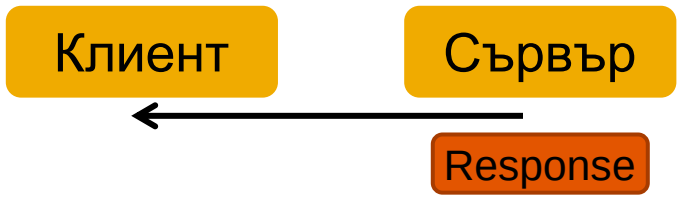


# Мрежова комуникация с Java

## Клиент-сървър комуникация (3)

Обработка на заявката и връщане на отговор:

```
try {
    //Process request
    //Send response
    PrintWriter pw = new PrintWriter(s.getOutputStream());
    pw.println("Hello World");
    pw.flush();
    pw.close();
    is.close();
    s.close();
} catch (IOException e) {
}
```

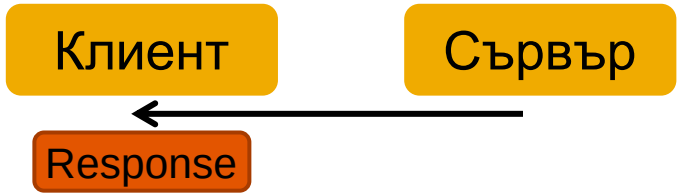


# Мрежова комуникация с Java

## Клиент-сървър комуникация (4)

Прочитане на отговора (**response**) от сървъра :

```
try {
    ...
    BufferedReader br = new BufferedReader(
        new InputStreamReader(s.getInputStream()));
    String line;
    while ((line = br.readLine()) != null) {
        System.out.println(line);
    }
    pw.close();
    br.close();
    s.close();
} catch (IOException e) {
}
```



# Мрежова комуникация с Java

## Пакетът java.nio (1)

Въведен от JDK 1.4.

Позволява асинхронни входно-изходни (I/O) операции.

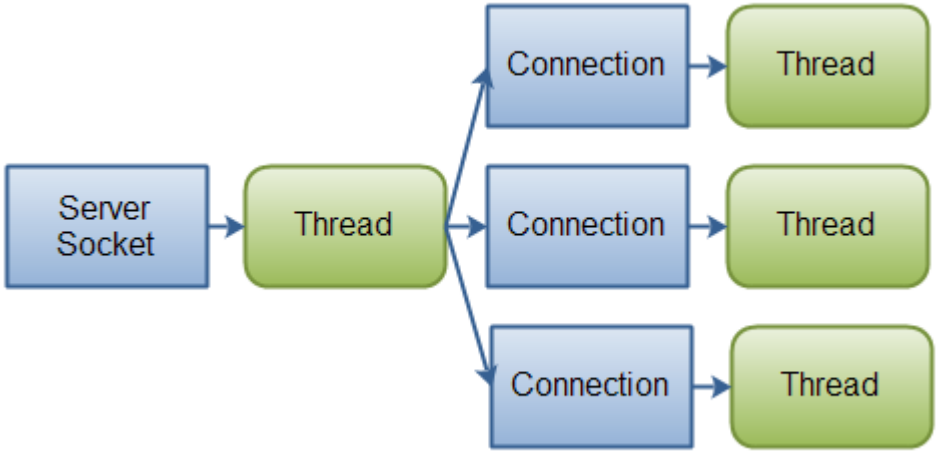
Намалява генерирането на боклук, чрез използването на буфери с директна памет (DMA), при които при писането и четенето в сокетите не се извършва копиране на данни.

Основни обекти:

- **Канали (Channels)** – Подобни на потоците (Stream). Представляват една връзка, като от тях може да се чете и да се записва. Основните класове: **FileChannel, DatagramChannel, SocketChannel, ServerSocketChannel**.
- **Буфери (Buffers)** – Представляват блок от паметта в която може да записваш данни. Използват се за четенето и запис в NIO канали (channels).
- **Селектор (Selector)** – Компонент в който се регистрират канали и може да обработва повече от един канал в една нишка.

# Мрежова комуникация с Java

## Пакетът java.nio (2)



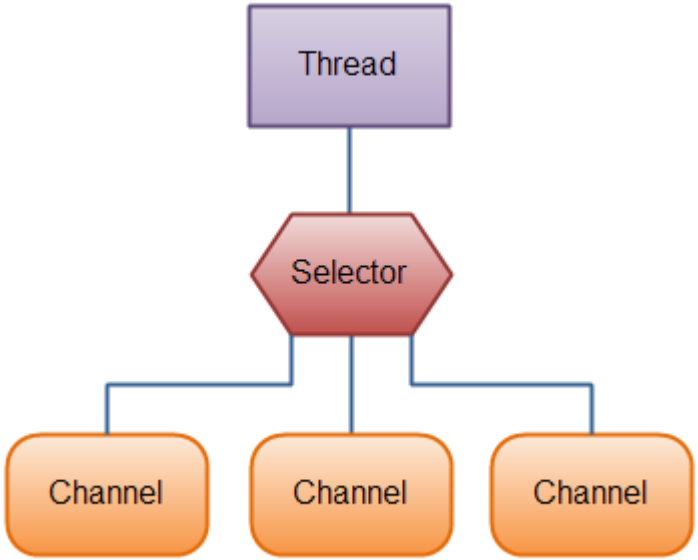

За класовете от пакета java.net е необходимо да има една нишка за всяка връзка (connection).

Синхронни (блокиращи) операции.

Нишка селектор (Selector), която позволява обработването на няколко канала от една нишка.

Намалява броя на нишките като премахва нуждата от отделна нишка за всяка връзка (connection).

Асинхронни (неблокиращи) операции.



# Мрежова комуникация с Java

## java.nio.channels.ServerSocketChannel

ServerSocketChannel е канал (java.nio.Channel), който може да слуша за входящи TCP повиквания, точно както ServerSocket.

```
ServerSocketChannel serverSocketChannel = ServerSocketChannel.open();
serverSocketChannel.socket().bind(new InetSocketAddress(9999));

while(true){
    SocketChannel socketChannel = serverSocketChannel.accept();
    ...
}
```

SocketChannel  представлява една TCP връзка. За да се използва асинхронно трябва да се регистрира в селектор.

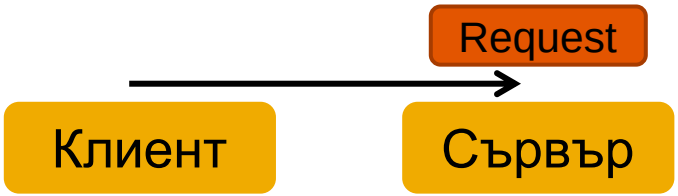
```
socketChannel.register(selector, SelectionKey.OP_READ);
```

# Мрежова комуникация с Java

## java.nio.channels.SocketChannel - Четене

Четене на няколко канала от един селектор.

```
while (true) {
    int readyChannels = selector.select();
    if (readyChannels == 0) continue;

    Set<SelectionKey> selectedKeys = selector.selectedKeys();
    Iterator<SelectionKey> keyIterator = selectedKeys.iterator();
    while (keyIterator.hasNext()) {
        SelectionKey key = keyIterator.next();
        if (key.isReadable()) {
            //A channel is ready for reading
        }
        keyIterator.remove();
    }
}
```

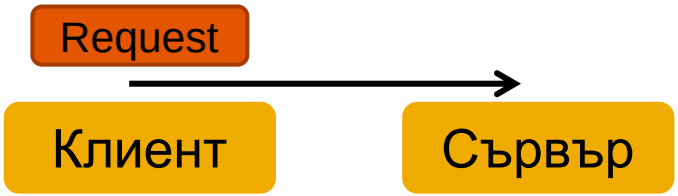


# Мрежова комуникация с Java

## java.nio.channels.SocketChannel - Запис

Използваме SocketChannel канал (channel) и за изпращане на данни по TCP връзката (connection).

```
socketChannel = SocketChannel.open();
socketChannel.connect(new InetSocketAddress("127.0.0.1", 9999));

String newData = "Current time: " + System.currentTimeMillis();
ByteBuffer buf = ByteBuffer.allocate(48);
buf.clear();
buf.put(newData.getBytes());
buf.flip();
while(buf.hasRemaining()) {
    socketChannel.write(buf);
}
```



# Мрежова комуникация с Java
## URL (1)

URL е съкращение от Uniform Resource Locator. Представлява адрес на ресурс (HTML страница, текстов документ, видео, и т.н.) в интернет.

**http://example.com:8080/pages/help.html#anchor**

Java предоставя класа java.net.URL за създаването на абсолютни или относителни адреси.

Може да използвате един от наличните конструктори, за да създадете URL обект.

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

new URL("http", "example.com", "/pages/page1.html");

new URL("http", "example.com", 80, "pages/page1.html");

try {

    URL myURL = new URL(...);

} catch (MalformedURLException e) {//Exception handler code here…}
```

# Мрежова комуникация с Java
## URL (2)

Класа URL предоставя методи за извличане на информация за URL обекта:

- getProtocol - връща идентификатор за протокол.
- getAuthority - връща “authority” компонента.
- getHost – връща името на хоста.
- getPort – връща номера на порта.
- getPath – връща пътя.
- getQuery – връща заявката.
- getFile – връща името на файла.
- getRef – връща референцията.

Пример:

**http://example.com:80/tutorial/index.html?name=networking#DOWNLOADING**

**protocol** = http
**authority** = example.com:80
**host** = example.com
**port** = 80
**path** = /tutorial/index.html
**query** = name=networking

**filename** = /tutorial/index.html?name=networking
**ref** = DOWNLOADING

# Мрежова комуникация с Java
## URL (3)

След като се създаде URL обект, може да се използва метода openStream(), за да се получите stream, от който може да се прочете съдържанието на ресурса.

```
try {
    URL sap = new URL("http://www.sap.com/");
    BufferedReader in = new BufferedReader(
    new InputStreamReader(sap.openStream()));

    String inputLine;
    while ((inputLine = in.readLine()) != null) {
        System.out.println(inputLine);
    }
    in.close();
} catch (MalformedURLException e) {
} catch (IOException e) {}
```

# Използвана литература

- Networking tutorial
  - http://docs.oracle.com/javase/tutorial/networking/TOC.html
- IO tutorial
  - http://docs.oracle.com/javase/tutorial/essential/io/
- NIO tutorial
  - http://tutorials.jenkov.com/java-nio/index.html