

A background image showing a complex network of interconnected nodes and lines, representing a network topology, set against a blue gradient.

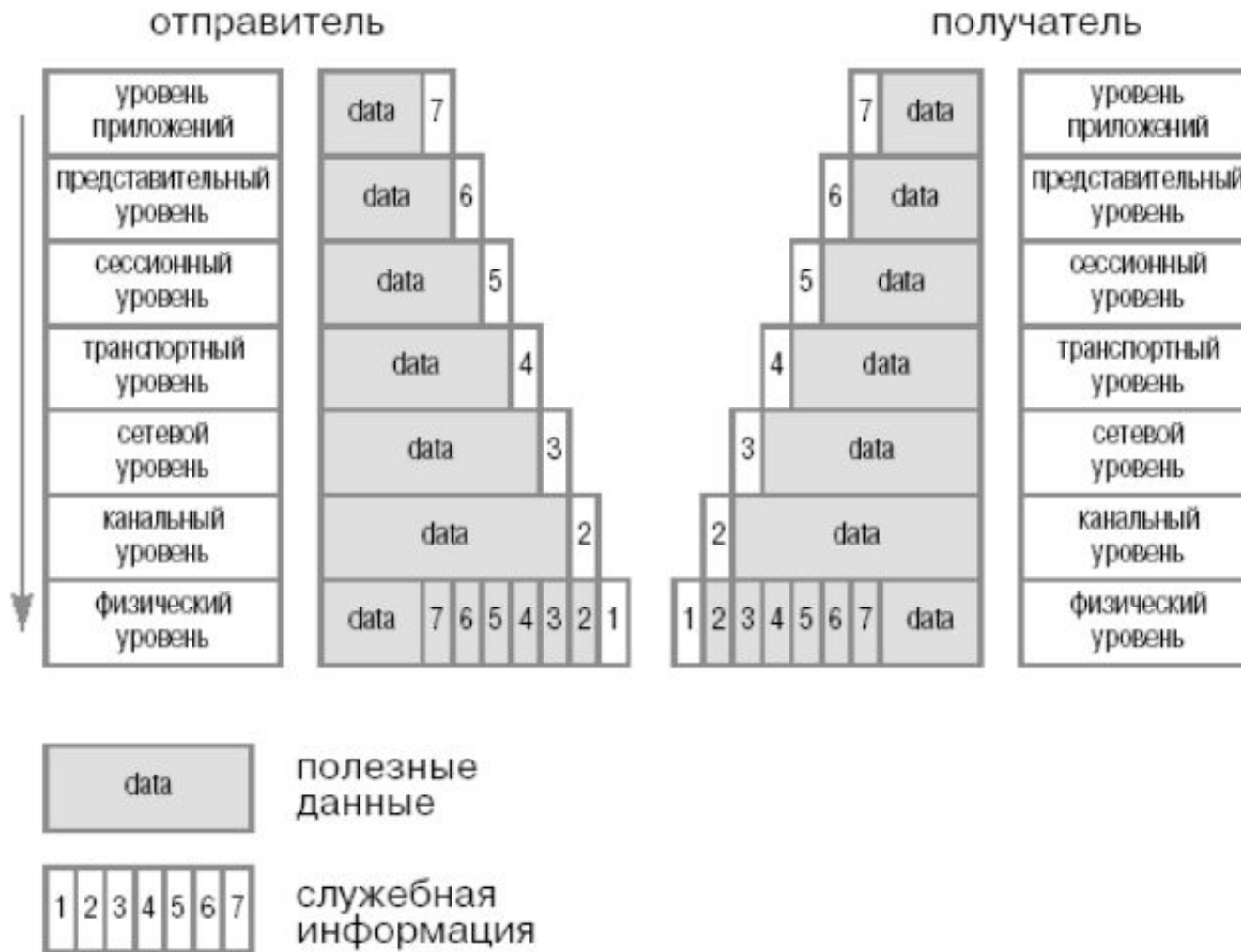
Лекция 11. Работа с сетями. Пакет java.net.

NetCracker®

Модель OSI

7	Прикладной уровень (уровень приложений)	напр. HTTP , SMTP , SNMP , FTP , Telnet , SMB , NFS , RTSP , BGP
6	Представительский уровень	напр. XDR , ASN.1 , AFP
5	Сеансовый уровень	напр. TLS , SSL , RPC , NetBIOS , ASP
4	Транспортный уровень	напр. TCP , UDP , RTP , SCTP , SPX , ATP , DCCP , GRE
3	Сетевой уровень	напр. IP , ICMP , IGMP , CLNP , OSPF , RIP , IPX , DDP
2	Канальный уровень	напр. Ethernet , Token ring , PPP , HDLC , X.25 , Frame relay , ISDN , ATM , MPLS , Wi-Fi , ARP , RARP
1	Физический уровень	напр. электрические провода , радиосвязь , оптоволоконные провода

Модель OSI



Transmission Control Protocol (TCP)

TCP – основанный на соединениях протокол, обеспечивающий надёжную передачу данных между двумя компьютерами с сохранением порядка данных.

Используется в HTTP, FTP, Telnet и др.

User Datagram Protocol (UDP)

UDP – не основанный на соединениях протокол, реализующий пересылку независимых пакетов данных, называемых дейтаграммами, от одного компьютера к другому без гарантии их доставки. Если данные были некорректно доставлены, или вообще часть пакетов потерялась - UDP не позволяет их восстановить. Запрос на получение данных должен будет выполнен заново.

Основные характеристики TCP и UDP

TCP	UDP
Для работы устанавливает соединение	Работает без соединений
Гарантированная доставка данных	Гарантий доставки нет
Разбивает исходное сообщение на сегменты	Передаёт сообщения целиком в виде дейтаграмм
На стороне получателя сообщение заново собирается из сегментов	Принимаемые сообщения не объединяются
Пересылает заново потерянные сегменты	Подтверждений о доставке нет
Контролирует поток сегментов	Никакого контроля потока дейтаграмм нет

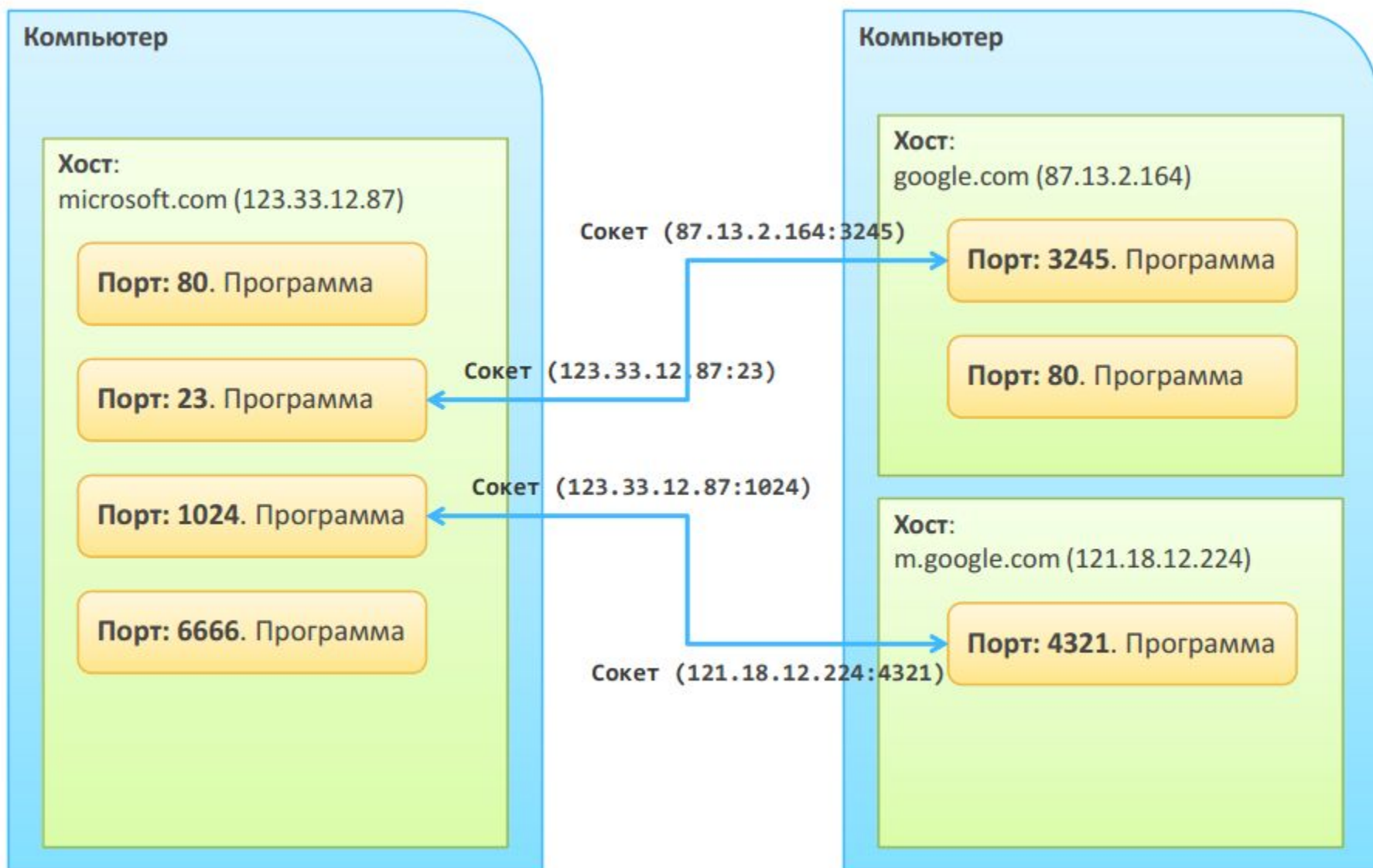
Модель «клиент-сервер»

- Приложение **сервер** инициализируется при запуске и далее бездействует, ожидая поступление запроса от клиента.
- Процесс **клиент** посылает запрос на установление соединения с сервером, требуя выполнить для него определенную функцию.

Понятие порта

- Компьютер (обычно) имеет только одно физическое соединение с сетью.
- Соединение описывается, например, IP-адресом.
- IP адреса не достаточно для уникальной идентификации сервера, так как многие сервера могут существовать на одной машине. Каждая IP машина также содержит порты, и когда вы устанавливаете клиента или сервер, вы должны выбрать порт, через который и клиент, и сервер согласны соединиться.
- **Порт** - это не физическое расположение в машине, а программная абстракция.
- Сокет привязывается к порту.
- Порт описывается 16-битным числом.
- Порты 0-1023 зарезервированы.

Понятие сокета



Пакет java.net

- **Адресация**
 - URI, URL, NetworkInterface, InterfaceAddress, InetAddress, IDN
- **Установление TCP соединения**
 - ServerSocket, Socket
- **Передача/приём дейтаграмм через UDP**
 - DatagramPacket, DatagramSocket, MulticastSocket
- **Обнаружение/идентификация сетевых ресурсов**
 - URI, URL, URLConnection, URLStreamHandler
- **Безопасность: авторизация/права доступа.**
 - Authenticator, PasswordAuthentication

Класс Socket

Чаще всего для создания сокетов в клиентских приложениях вы будете использовать один из двух конструкторов:

```
Socket(String host, int port)
```

```
Socket(InetAddress address, int port)
```

В классе Socket определена еще одна пара конструкторов, которая, однако не рекомендуется для использования:

```
Socket(String host, int port, boolean stream)
```

```
Socket(InetAddress address, int port, boolean stream)
```

В этих конструкторах последний параметр определяет тип сокета. Если этот параметр равен true, создается потоковый сокет, а если false - датаграммный. Заметим, что для работы с датаграммными сокетами следует использовать класс DatagramSocket.

Методы класса Socket

Методы **getInputStream** и **getOutputStream**, предназначены для создания входного и выходного потока, соответственно:

- **getInputStream()**
- **getOutputStream()**

Эти потоки связаны с сокетом и должны быть использованы для передачи данных по каналу связи.

Методы **getInetAddress** и **getPort** позволяют определить адрес IP и номер порта, связанные с данным сокетом (для удаленного узла):

- **getInetAddress()**
- **getPort()**

Метод **getLocalPort** возвращает для данного сокета номер локального порта:

- **getLocalPort()**

После того как работа с сокетом завершена, его необходимо закрыть методом **close**:

- **close()**

Порядок работы с сокетом клиента

1. открытие сокета;
2. открытие потока ввода и/или потока вывода для сокета;
3. чтение и запись в потоки согласно установленному протоколу общения с сервером;
4. закрытие потока ввода-вывода верхнего уровня (если таковые создавались для обертки низкоуровневых потоков);
5. закрытия сокета.

Пример клиента

```
Socket s = new Socket("localhost", 3456);
try {
    InputStream is = s.getInputStream();
    try {
        System.out.println("Read: " + is.read());
    }
    finally {
        is.close();
    }
}
finally {
    s.close();
}
```

ServerSocket

Реализует серверный сокет и его функции;

Конструкторы:

- `ServerSocket()`
- `ServerSocket(int port)`
- `ServerSocket(int port, int backlog)`

Методы:

- `close()`
- `accept()`
- `bind(SocketAddress endpoint)`
- и прочие...

Пример сервера

```
ServerSocket ss = new ServerSocket(3456);
try {
    System.out.println("Waiting...");
    Socket client = ss.accept();
    try {
        System.out.println("Connected");
        OutputStream out = client.getOutputStream();
        try {
            out.write(10);
        }
        finally {
            out.close();
        }
    }
    finally {
        client.close();
    }
}
finally {
    ss.close();
}
```

Дейтаграммы

Дейтаграмма – независимое, самодостаточное сообщение, посылаемое по сети, чья доставка, время (порядок) доставки и содержимое не гарантируются.

- Могут использоваться как для адресной так и для широковещательной рассылки.
- Для посылки дейтаграмм отправитель и получатель создают сокет дейтаграммного типа.

DatagramSocketclass

Дейтаграммы представлены классом DatagramSocket.

В классе три конструктора:

- - DatagramSocket()— создаваемый сокет присоединяется к любому свободному порту на локальной машине;
- - DatagramSocket(int port)— создаваемый сокет присоединяется к

порту port на локальной машине;

- - DatagramSocket(int port, InetAddress addr)— создаваемый сокет присоединяется к порту port; аргумент addr— один из адресов локальной машины.

DatagramSocket

После открытия сокетов начинается обмен датаграммами. Они представляются экземплярами класса **DatagramPacket**. При отсылке сообщения применяется следующий конструктор:

```
DatagramPacket(byte[] buf, int length, InetAddress address, int port)
```

Массив содержит данные для отправки (созданный пакет будет иметь длину равную **length**), а адрес и порт указывают получателя пакета. После этого вызывается метод **send()** класса **DatagramSocket**.

Пример отправки и приёма дейтаграмм

•••• Отправка дейтаграмм:

```
DatagramSocket s = new DatagramSocket();
try {
    byte data[] = {1, 2, 3};
    InetAddress addr =
        InetAddress.getBy_name("localhost");
    DatagramPacket p =
        new DatagramPacket(data, 3, addr, 3456);
    s.send(p);
    System.out.println("Datagram sent");
}
catch (SocketException e) { ... }
catch (UnknownHostException e) { ... }
catch (IOException e) { ... }
finally {
    s.close();
}
```

•••• Получение дейтаграмм:

```
DatagramSocket s = new DatagramSocket(3456);
try {
    byte data[] = new byte[3];
    DatagramPacket p =
        new DatagramPacket(data, 3);
    System.out.println("Waiting...");
    s.receive(p);
    System.out.println("Datagram received: " +
        data[0] + ", " + data[1] + ", " +
        data[2]);
}
catch (SocketException e) { ... }
catch (IOException e) { ... }
finally {
    s.close();
}
```

URL-соединения

Для работы с ресурсами, заданными своими адресами URL, в библиотеке Java имеется очень удобный и мощный класс – **URL**.

Конструкторы URL:

- `URL(String spec);`
- `URL(String protocol, String host, int port, String file)`
- `URL(String protocol, String host, String file)`
- `URL(URL context, String spec)`

Методы:

- `getContent()`
- `openConnection()`
- `openStream()`
- `getHost(), getFile(), getProtocol()`

Используемая литература:

- Курс лекций МФТИ «Программирование на Java»
- <http://www.intuit.ru/department/pl/javapl/16/>

Thank you!

••••