

# СЕТЕВЫЕ ПРОГРАММЫ



# Сетевое программирование

java.net

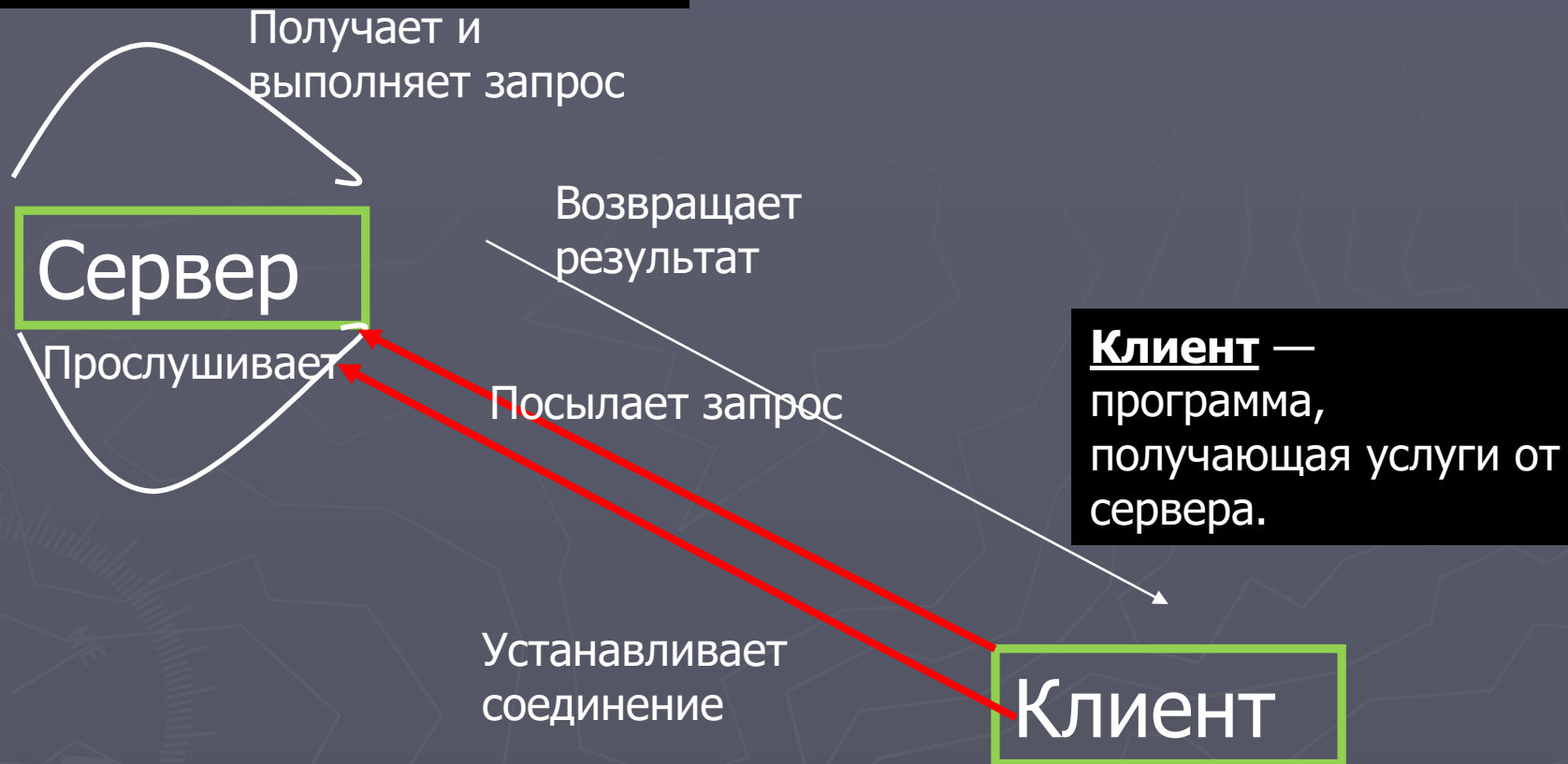
1. установка сетевого соединения
2. передача запроса и сообщения  
(многопоточность)

Сетевые приложения - веб-браузер, e-mail, сетевые новости, передача файлов и т.д.

Используются сокеты, порты, протоколы  
TCP/IP, UDP

**Программа-сервер** — обычно устанавливается на удаленном компьютере и предоставляет услуги другим программам - клиентам

# клиент/сервер



Запросы и сообщения - записи, структура которых определяется используемыми протоколами.

# Стек протоколов TCP/IP

## Transmission Control Protocol/Internet Protocol

это комбинация протоколов, которые работают в сети одновременно и обеспечивают следующие операции с данными:

- Подготовку
- Передачу
- Прием

Стек протоколов TCP/IP включает в себя четыре уровня:

- ▶ прикладной уровень (application layer),
- ▶ транспортный уровень (transport layer),
- ▶ сетевой уровень (Internet layer),
- ▶ канальный уровень (link layer).

# OSI

# TCP/IP

7	Прикладной	Прикладной						I
6	Представления	Telnet	FTP	SMTP	HTTP	RIP	SNMP	
5	Сеансовый							
4	Транспортный	Транспортный						II
		TCP			UDP			
3	Сетевой	Межсетевой						III
		IP						
2	Канальный	Сетевой интерфейс Не регламентируется: Ethernet, TokenRing, X.25, ATM и т.д. ....						IV
1	Физический							

обеспечивает доступ к сервисам других уровней и определяют протоколы, по которым приложения могут обмениваться данными

Предоставляет прикладному уровню сеансовые коммуникационные службы.

IP- маршрутизируемый протокол, отвечающий за IP-адресацию, маршрутизацию, фрагментацию и восстановление пакетов.

отвечает за организацию взаимодействия с технологиями сетей, входящими в составную

- HTTP(s) — Hypertext Transfer Protocol (WWW);
- NNTP — Network News Transfer Protocol (группы новостей);
- SMTP — Simple Mail Transfer Protocol (посылка почты);
- POP3 — Post Office Protocol (чтение почты с сервера);
- FTP — File Transfer Protocol (протокол передачи файлов).

Каждый компьютер из подключенных к сети по протоколу TCP/IP имеет уникальный IP-адрес, используемый для идентификации и установки соединения.

IPv4

32-битовое число-217.21.43.10

IPv6

128-битовое число- 1170:0:0:0:7:771:100A:214B.

Если в качестве сервера используется этот же компьютер без сетевого подключения, в качестве IP-адреса указывается 127.0.0.1 или localhost

- Для явной идентификации услуг к IP-адресу присоединяется номер порта 217.21.43.10:443

указывает конкретное место соединения на указанном компьютере и предоставляет определенную услугу

- URL (Universal Resource Locator) состоит из двух частей — префикса протокола (http, https, ftp и т. д.) и URI (Universal Resource Identifier).

# Определить IP-адрес

## ► java.net.InetAddress (Inet4Address и Inet6Address)

```
InetAddress currentIP = null;  
currentIP = InetAddress.getLocalHost();
```

```
getByName(String host) //InetAddress,  
    //содержащий IP-адрес по имени компьютера  
getByAddress(byte[] addr) //InetAddress,  
    //содержащий имя компьютера, по IP-адресу  
getAllByName(String host) //массив объектов  
    // класса InetAddress  
UnknownHostException // исключения
```

```
boolean isReachable(int timeout)
```

Проверить доступ к компьютеру

## получить IP-адрес текущего компьютера и IP-адрес из имени домена

```
InetAddress currentIP = null;
InetAddress belstuIP = null;
try {
    currentIP = InetAddress.getLocalHost();
    // вывод IP-адреса локального компьютера

    System.out.println("IP -> " + currentIP.getHostAddress(););

    belstuIP = InetAddress.getByName("www.belstu.by");
    // вывод IP-адреса БГУ www.belstu.by

    System.out.println("BSTU -> " + belstuIP.getHostAddress(););

} catch (UnknownHostException e) {
    // если не удастся найти IP
    System.err.println("адрес недоступен " + e);
}
```

"C:\Program ...

IP -> 192.168.0.3

BSTU -> 108.162.203.87



```
//Определение доступа
// задание IP-адреса в виде массива
byte ip[] = {(byte)123, (byte)162, (byte)204, (byte)87};
try {
    InetAddress addr = InetAddress.getByAddress("Unknow", ip);

    System.out.println(addr.getHostName()
        + "-> соединение:" + addr.isReachable(100));

} catch (UnknownHostException e) {
    System.err.println("адрес недоступен " + e);
} catch (IOException e) {
    System.err.println("ошибка " + e);
}
```

10.10.10.10 -> 100.104.203.01

Unknow-> соединение:false

Default F

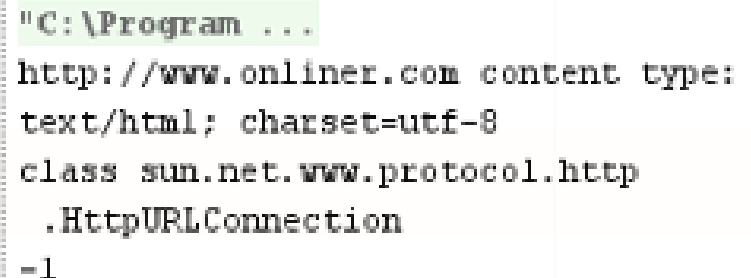
# Пример - чтение html файла по url

```
public static void main(String[] args) {
    URL tut = null;
    String urlName = "http://www.tut.by";
    try {
        tut = new URL(urlName);
    } catch (MalformedURLException e) {
        // некорректно заданы протокол, доменное имя или путь к файлу
        e.printStackTrace();
    }
    if (tut == null) {
        throw new RuntimeException();
    }
    try (BufferedReader d = new BufferedReader
        (new InputStreamReader(bsu.openStream())))
    {
        String line = "";
        while ((line = d.readLine()) != null) {
            System.out.println(line);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

```
"C:\Program ...
<!DOCTYPE html>
<!--[if lt IE 7]><html class="no-js
ie6 oldie desktop lang-rus"
xmlns:fb="http://ogp.me/ns/fb#"
lang="ru"><![endif]-->
<!--[if IE 7]><html class="no-js
ie7 oldie desktop lang-rus"
xmlns:fb="http://ogp.me/ns/fb#"
lang="ru"><![endif]-->
<!--[if IE 8]><html class="no-js
ie8 desktop lang-rus"
xmlns:fb="http://ogp.me/ns/fb#"
lang="ru"><![endif]-->
<!--[if gt IE 8]><!--><html
class="no-js desktop lang-rus"
xmlns:fb="http://ogp.me/ns/fb#"
lang="ru"><!--<![endif]-->
<head>
  <meta charset="utf-8" />
  <meta http-equiv="x-dns-prefetch
-control" content="on" />
  <link rel="dns-prefetch"
href="http://img.tyt.by/" />
  <link rel="dns-prefetch"
href="http://news.tut.by/" />
  <link rel="dns-prefetch"
href="http://kupi.tut.by/" />
  <link rel="dns-prefetch"
href="http://img.dir.tut.by/" />
  <link rel="dns-prefetch"
href="http://reklama.tut.by/" />
  <link rel="dns-prefetch"
```

# Пример – извлечение информации о соединении

```
public static void main(String[] args) {
    String urlName = "http://www.onliner.com";
    int timeout = 10_000;
    URL url;
    try {
        url = new URL(urlName);
        final URLConnection connection = url.openConnection();
        // установка таймута на соединение
        connection.setConnectTimeout(timeout);
        System.out.println(urlName + " content type: " +
            "\n" + connection.getContentType() +
            "\n" + connection.getClass() +
            "\n" + connection.getContentLength());
        // продолжение извлечения информации из соединения
    } catch (MalformedURLException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```



```
"C:\Program ...
http://www.onliner.com content type:
text/html; charset=utf-8
class sun.net.www.protocol.http
.HttpURLConnection
-1
```

# Сокетные соединения по TCP/IP

**Сокеты** (сетевые разъемы) — логическое понятие, соответствующее разъемам, к которым подключены сетевые компьютеры и через которые осуществляется дуплексная поточная передача данных между компьютерами.

определяется IP-адресом и номером порта

для идентификации компьютера

для идентификации процесса

## ► Сокетное протоколо-ориентированное соединение по протоколу TCP/IP

- 1) Сервер создает сокет, который будет использоваться для связи с клиентом
- 2) Сервер прослушивает сообщение и ждет, пока клиент не свяжется с ним.
- 3) Первое сообщение, посылаемое клиентом на сервер, содержит сокет клиента
- 4) Сервер посылает сокет клиенту с первым сообщением
- 5) Устанавливается коммуникационное соединение.

соединение с сокетом имеет гораздо меньше накладных расходов, чем любой стандартный протокол.

обеспечивает двусторонний поток байт, в котором данные перемещаются одновременно в обоих направлениях, в отличие от протоколов, основанных на модели запрос-ответ.

```
Socket socketClient = new Socket ("128.34.56.3", 8030);
```

## Установка соединения на стороне сервера

```
ServerSocket server = new ServerSocket (8030);  
Socket socketServ = server.accept();  
//ожидает клиента и возвращает его сокет
```

## Установка соединения сервером

```
getInputStream() //получать данные из потока ввода  
getOutputStream() //записывать данные в поток вывода  
PrintStream //трактовка поток как выходного файла
```

## Обмен данным

```
close()
```

## Закрытие сокета

## ► JVM (сервер)

### 1) Стартуем сервер на к-н порту

```
new ServerSocket (3000) ;
```

### 2) Устанавливаем в режим прослушивания `accept () ;`

### 3) Обработка запроса и возврат результата

## ► JVM (клиент)

### 1) Соединение с сокетом сервера

```
socketClient = new Socket ("128.34.56.3", 3000) ;
```

### 2) Получение ссылки на поток сервера

```
out = socketClient.getOutputStream () ;
```

### 3) Посылка запроса в поток

### 4) Обработка результатов



# Пример – сервер деканат, по запросу (номер группы – клиент) вернет лучшего студента

## Сервер

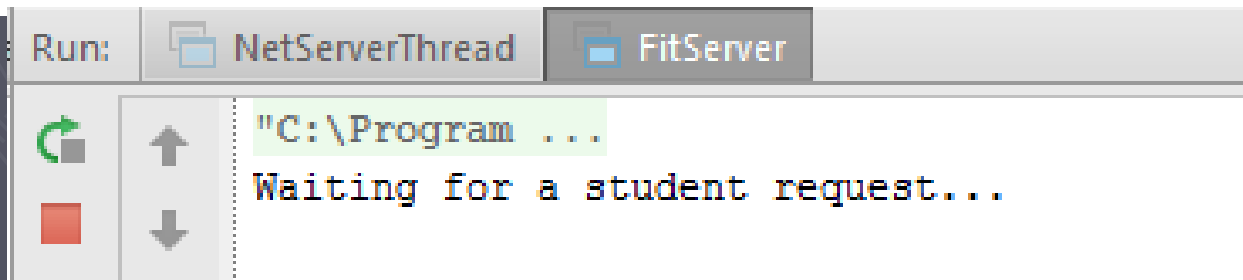
```
public class FitServer {  
    public static void main(java.lang.String[] args) {  
        ServerSocket serverSocket = null;  
        Socket client = null;  
  
        BufferedReader inbound = null;  
        OutputStream outbound = null;  
  
        List<String> gr3 = new ArrayList<String>();  
        gr3.add("Mihail");  
        gr3.add("Alexandr");  
        gr3.add("Lena");  
        gr3.add("Nikita");  
        try  
        {
```

```
// Создаем server socket
serverSocket = new ServerSocket(3000);

System.out.println("Waiting for a student request...");
while (true)
{
    // Ждем запрос
    client = serverSocket.accept();

    // Получаем поток
    inbound=new BufferedReader(new
        InputStreamReader(client.getInputStream()));
    outbound = client.getOutputStream();

    String symbol = inbound.readLine();
```



```
//Генерируем студента
```

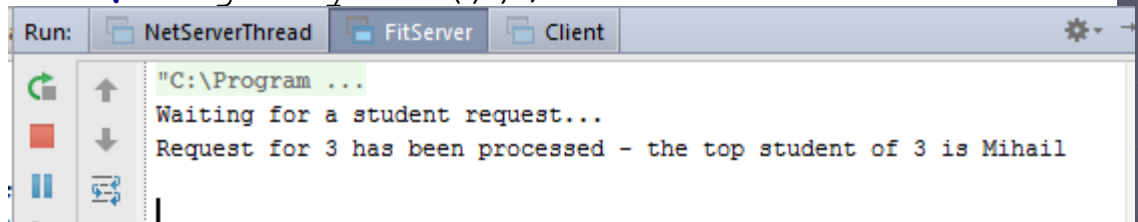
```
String student= gr4.get((int) (Math.random()*3));  
outbound.write(("\\n The top student of " + symbol +  
        " is " + student + "\\n").getBytes());
```

```
System.out.println("Request for " + symbol +  
        " has been processed - the top student of "  
        + symbol+ " is " + student + "\\n" );  
outbound.write("End\\n".getBytes());
```

```
}
```

```
}
```

```
catch (IOException ioe) {  
    System.out.println("Error in Server: " + ioe);  
} finally{  
    try{  
        inbound.close();  
        outbound.close();  
    } catch (Exception e) {  
        System.out.println("FitServer:  
        can't close streams" + e.getMessage());  
    }  
}  
  
}  
  
}
```



# Клиент

```
public class Client {  
    public static void main(java.lang.String[] args) {  
  
        Socket clientSocket = null;  
        try{  
            // Открыть клиентское socket connection  
            clientSocket = new Socket("localhost", 3000);  
  
            System.out.println("Client: " + clientSocket);  
        } catch (UnknownHostException uhe) {  
            System.out.println("UnknownHostException: " + uhe);  
        } catch (IOException ioe) {  
            System.err.println("IOException: " + ioe);  
        }  
    }  
}
```

Порт

80 - для HTTP серверов;

443 – для HTTPS;

21 - для FTP коммуникаций;

389 – для LDAP серверов

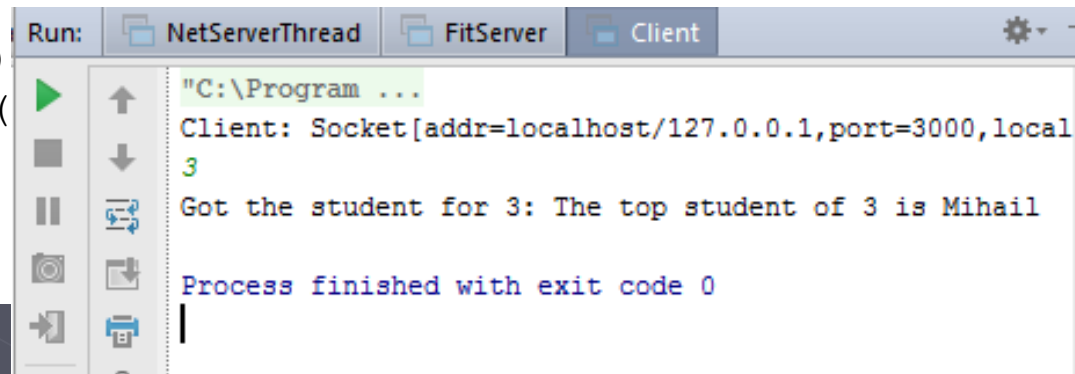
```

try (OutputStream outbound = clientSocket.getOutputStream();
     BufferedReader inbound = new BufferedReader(new
         InputStreamReader(clientSocket.getInputStream())); )
{
    Scanner scan = new Scanner(System.in);
    int group = scan.nextInt();
    // Послать символ серверу
    outbound.write((group+"\n").getBytes());

    String topStudent;
    while (true) {
        topStudent = inbound.readLine();
        if (topStudent.length() == 0) continue;

        if (topStudent.equals("End")) {
            break;
        }
        System.out.println("Got the student for " +
            group + ":" + topStudent);
    }
} catch (IOException ioe) {
    ioe.printStackTrace();
}
}

```



```

Run: NetServerThread FitServer Client
"C:\Program ...
Client: Socket[addr=localhost/127.0.0.1,port=3000,local
3
Got the student for 3: The top student of 3 is Mihail
Process finished with exit code 0

```

# МНОГОПОТОЧНОСТЬ - сервер

```
class ServerThread extends Thread {  
    private PrintStream os; // передача  
    private BufferedReader is; // прием  
    private InetAddress addr; // адрес клиента  
    public ServerThread(Socket s) throws IOException {  
        os = new PrintStream(s.getOutputStream());  
        is = new BufferedReader(new InputStreamReader(s.getInputStream()));  
        addr = s.getInetAddress();  
    }  
    public void run() {  
        int i = 0;  
        String str;  
        try {  
            while ((str = is.readLine()) != null) {  
                if ("PING".equals(str)) {  
                    os.println("PONG " + ++i);  
                }  
                System.out.println("PING-PONG " + i + " with " +  
addr.getHostName());  
            }  
        } catch (IOException e) {  
            // если клиент не отвечает, соединение с ним разрывается  
            System.err.println("Disconnect");  
        }  
        finally {  
            disconnect(); // уничтожение потока  
        }  
    }  
}
```

Сервер должен поддерживать  
многопоточность для обработки  
несколько соединений одновременно.

Сервер содержит цикл,  
ожидающий нового  
клиентского соединения.  
Каждый раз, когда клиент  
просит соединения,  
сервер создает новый  
поток.

```
public void disconnect() {  
    try {  
        if (os != null) {  
            os.close();  
        }  
        if (is != null) {  
            is.close();  
        }  
        System.out.println(addr.getHostName() + " disconnecting");  
    } catch (IOException e) {  
        e.printStackTrace();  
    } finally {  
        this.interrupt();  
    }  
}  
}
```

```
public class NetServerThread {
    public static void main(String[] args) {
        try {
            ServerSocket server = new ServerSocket(7071);
            System.out.println("initialized");
            while (true) {
                // ожидание клиента
                Socket socket = server.accept();
                System.out.println(socket.getInetAddress().getHostName()
                    + " connected");
                /*
                * создание отдельного потока для обмена данными
                * с соединившимся клиентом
                */
                ServerThread thread = new ServerThread(socket);
                // запуск потока
                thread.start();
            }
        } catch (IOException e) {
            System.err.println(e);
        }
    }
}
```

ждет поступления запроса,  
приостанавливая  
выполнение серверной  
программы



# МНОГОПОТОЧНОСТЬ КЛИЕНТ

```
public class NetClient {
    public static void main(String[] args) {
        Socket socket = null;
        BufferedReader br = null;
        try {
            // установка соединения с сервером
            socket = new Socket(InetAddress.getLocalHost(), 7071);

            PrintStream ps =
                new PrintStream(socket.getOutputStream());
            br = new BufferedReader(new InputStreamReader(
                socket.getInputStream();));
            for (int i = 1; i <= 10; i++) {
                ps.println("PING");
                System.out.println(br.readLine());
                Thread.sleep(1_000);
            }
        }
    }
}
```

# Non- blocking sockets

## ► java.nio

### Non-blocking io

Параллельная работа - скорость работы выше.

### java.nio.buffer

java.nio.channels - это логические порталы, через которые осуществляется ввод/вывод данных, а буферы являются источниками или приёмниками этих переданных данных.

java.nio.channels.Selector - позволяют одному потоку выполнения мониторить несколько каналов ввода.

# Датаграммы и протокол UDP

## UDP (User Datagram Protocol)

- ▶ не устанавливает виртуального соединения и не гарантирует доставки данных

создает сокет присоединяется к любому свободному порту на локальной машине

```
DatagramSocket () //
```

создаваемый сокет присоединяется к порту port на локальной машине


```
DatagramSocket (int port)
```

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

```
DatagramPacket (byte[] buf, int length) //прием
```

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

```
DatagramSocket // и в роли клиента и сервера,  
  
    send(DatagramPacket pack) // отправить дейтаграмму  
                                упакованную в пакет  
    receive(DatagramPacket pack) // дожидается получения  
                                дейтаграммы и заносит ее в  
                                пакет  
connect(InetAddress addr, int port)  
    close();
```



При обмене дейтаграммами соединение обычно не устанавливается, дейтаграммы посылаются в расчете на то, что получатель ожидает их.  
Но можно установить соединение методом

## ► Недостатки

- Отправитель не узнает что сообщение не дошло – не гарантирует отправку
- Порядок не определен

## ► Достоинства

- Высокая скорость передачи (видео и аудио)

# Remote Method Invocation(RMI)

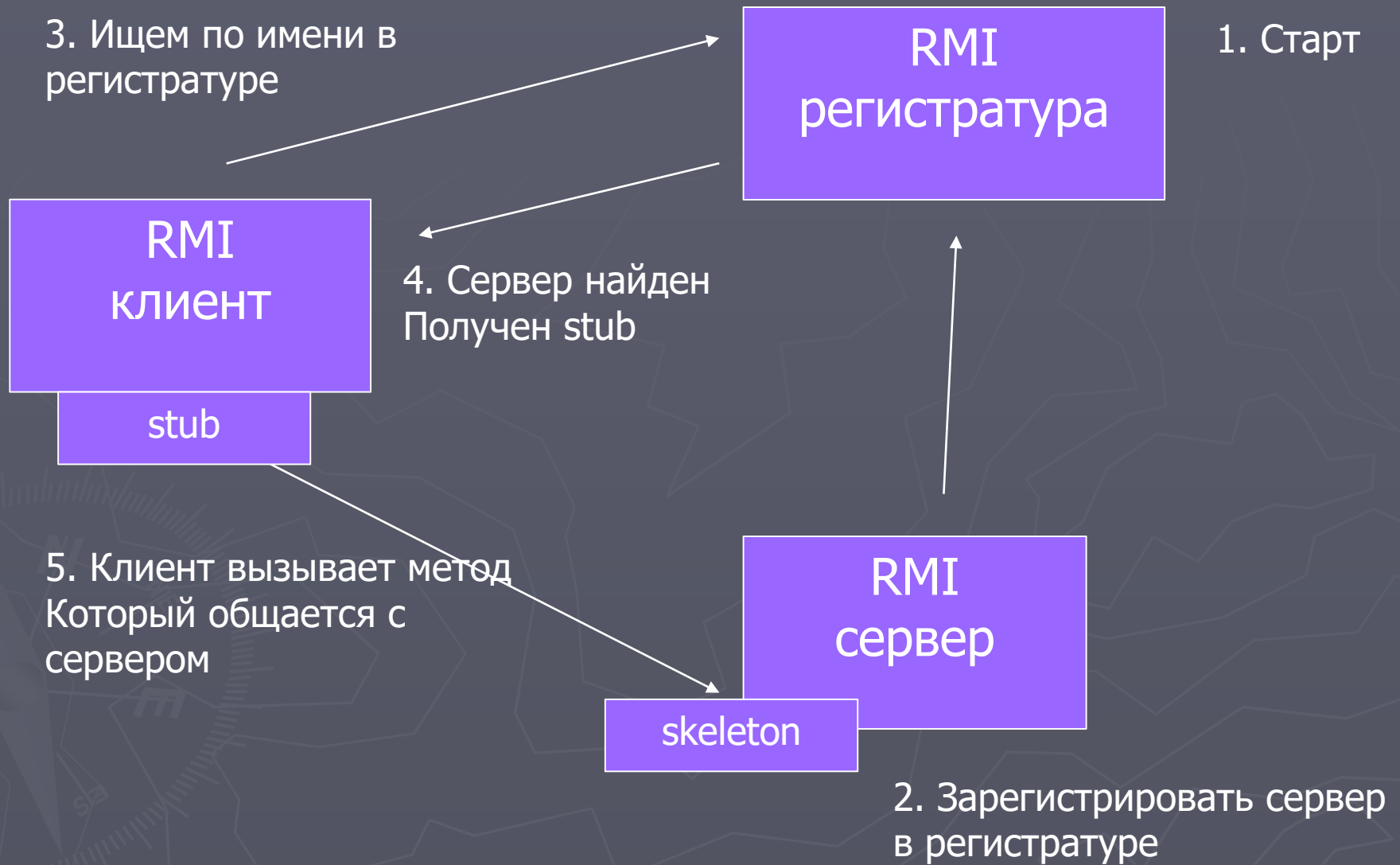
- ▶ Программный интерфейс, который позволяет вызывать метод удалённого объекта
- ▶ объекты располагаются на другой машине
- ▶ вы можете посылать сообщения этим удаленным объектам и получать результат, как будто они располагаются на вашей локальной машине.

- ▶ Создает регистратуру
- ▶ Выдает проху (stub) - передает запрос на сервер
- ▶ RMI –клиент ищет сервисы по имени
- ▶ RMI- сервер запускает регистратуру и регистрирует сервис по имени (или др.)

`java.rmi.registry.Registry lookup()`

номер порта по умолч. 1099

`rmi://<host_name>[:<name_port>]/<service_name>`





# Пример - деканат выдает оценки по списку

```
public interface DecanatServer extends Remote {  
  
    public String getMark(String symbol) throws  
        RemoteException;  
  
    public List<String> getStList() throws  
        RemoteException;  
  
}
```

```
public class FitServerImpl extends UnicastRemoteObject
    implements DecanatServer {

    private String mark=null;
    private ArrayList<String> stList = new ArrayList<String>();

    public FitServerImpl() throws RemoteException {
        super();
        LocateRegistry.createRegistry(1099);
        // Определение списка
        stList.add("GUZU");
        stList.add("SA");
        stList.add("SMOLIK");
        stList.add("SHATILO");
        stList.add("ZELINSKI");
    }
}
```

```
public String getMark(String symbol)
    throws RemoteException {

    mark=null;
    if(stList.indexOf(symbol.toUpperCase()) != -1) {
        // Генерация оценки

        mark = String.valueOf(((int) (Math.random()*10)));
    }

    return mark;
}

@Override
public List<String> getStList() throws RemoteException {
    return stList;
}

}
```

```
public class StartServer {  
  
    public static void main (String args[]) {  
  
        try {  
  
            // FitServerImpl - bind it  
            // регистрирует ITService  
            FitServerImpl shiman = new FitServerImpl();  
            Naming.rebind("rmi://pnv:1099/ITService", shiman);  
  
            System.out.println("<ITService> server is ready.");  
  
        } catch (MalformedURLException e1) {  
            System.out.println(e1.getMessage());  
        } catch (RemoteException ex) {  
            ex.printStackTrace();  
        }  
    }  
}
```

Требуется чтобы репозиторий был  
запущен как отдельный процесс  
на компьютере. Имя сервера  
репозитория rmiregistry:  
**start rmiregistry**

```
public class Client {  
  
    public static void main (String args[]) {  
  
        try {  
            DecanatServer myServer = (DecanatServer)  
                Naming.lookup("rmi://pnv:1099/ITService");  
  
            Scanner scan = new Scanner(System.in);  
            String student = scan.nextLine();  
            String bll = myServer.getMark(String.valueOf(student));  
            if (bll != null) {  
                System.out.println("The score of " + student +  
                    " is:" + bll);  
            }  
            else {  
                System.out.println("Invalid FIO. " +  
                    "Please use one of these:" +  
                    myServer.getStList().toString());  
            }  
        }  
    }  
}
```