



Занятие 1.13. Лекция

ОСНОВЫ STREAM API



Учебные вопросы

- 1. Вложенные и внутренние классы. Lambda-выражения.**
- 2. Java Stream API**
- 3. Работа с сетью в Java**



1. Вложенные и внутренние классы. Lambda-выражения

Класс в классе

Java позволяет создавать одни классы внутри других. Такие классы называют **вложенными (Nested)**.

Виды вложенных классов:

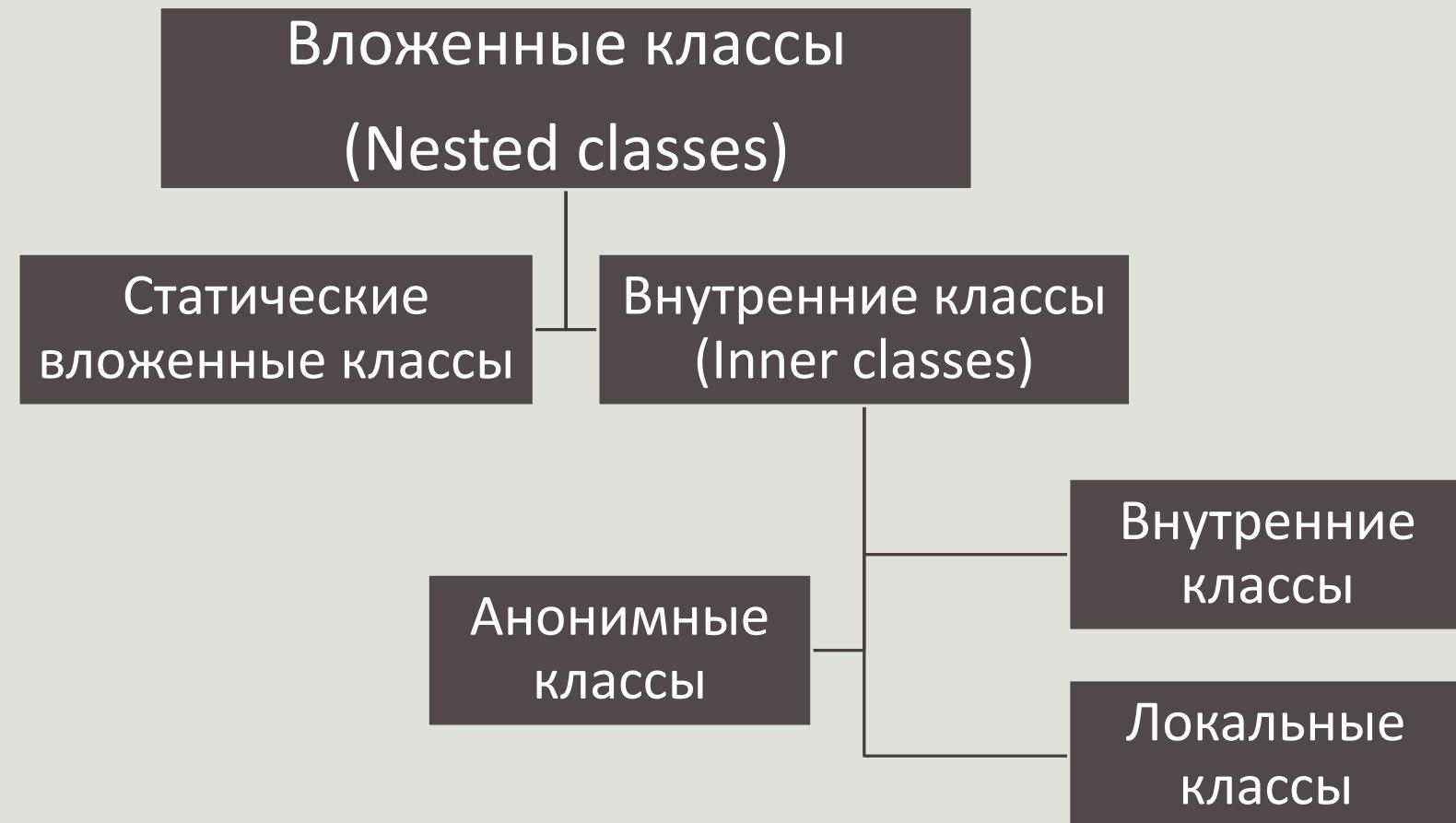
- Inner classes – внутренние классы
(нестатические вложенные классы)
- Static nested classes – статические вложенные классы



Вложенные классы

```
public class Practical8 {  
  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
  
    class InnerClass {  
  
    }  
  
    static class StaticNestedClass {  
  
    }  
}
```

Вложенные классы



Внутренние классы. Пример

```
public class Car {  
    private String brand;  
    private int year;  
  
    public void go() {  
        // какой-то код, описывающий движение автомобиля  
    }  
  
    public class Engine{  
        public void fuelCombustion(){  
            // какой-то код  
        }  
    }  
  
    public class Suspension{  
        // переменные и методы класса Suspension  
    }  
}
```

Внутренние классы. Пример

```
public static void main(String[] args) {  
    Engine en = new Engine();  
    en.fuelCombustion();  
}
```



```
public static void main(String[] args) {  
    Car car = new Car(brand: "AUDI", year: 2010);  
    Car.Engine eng = car.new Engine(); // создание экземпляра внутреннего класса  
    car.start(); // завели машину  
    eng.fuelCombustion(); // двигатель начал сжигать топливо  
    car.go(); // ПОЕХАЛИ!!  
}
```

Особенности внутренних классов

- ❑ Объект внутреннего класса не может существовать без объекта «внешнего» класса.
- ❑ У объекта внутреннего класса есть доступ к переменным «внешнего» класса (в том числе приватным).
- ❑ Внутренний класс может иметь привычные модификаторы доступа как и обычный класс

Локальные классы. Пример

```
public class PhoneNumberValidator {  
  
    public void validatePhoneNumber(String number) {  
        class PhoneNumber {  
            private String phoneNumber;  
            public PhoneNumber() {  
                this.phoneNumber = number;  
            }  
            public String getPhoneNumber() {  
                return phoneNumber;  
            }  
            //еще какие-то методы класса PhoneNumber  
        }  
        //...код проверки номера  
    }  
}
```

Анонимные классы

Анонимный класс (anonymous class) – это локальный класс без имени. Используется тогда, когда нужно переопределить метод класса или интерфейса. Класс одновременно объявляется и инициализируется.



Без использования анонимных классов. Пример

```
public class MyThread extends Thread{  
  
    @Override  
    public void run() {  
        // этот код выполнится в другом потоке  
        System.out.println("Hello, I am " + getName());  
    }  
}
```

```
public static void main(String[] args) {  
    Thread thr = new MyThread();  
    thr.start();  
}
```

Анонимные классы. Пример

```
public static void main(String[] args) {
    // создание и инициализация класса-наследника Thread
    Thread th = new Thread() {
        @Override
        public void run() {
            // этот код выполнится в другом потоке
            System.out.println("Hello, I am " + getName());
        }
    };
    th.start();
}
```

Особенности анонимных классов

- Не могут быть абстрактными или статическими
- Неявно являются финализированными (final)
- Всегда являются внутренними классами
- Могут не только переопределить методы класса наследника, но и содержать свои новые методы
- Могут быть объявлены не только в методе, но и внутри аргумента метода.

Анонимные классы. Использование в аргументах. Пример

```
public static void main(String[] args) {
    System.out.println("Hello! I am " + Thread.currentThread().getName());
    MyRunnable rn = new MyRunnable();
    Thread th = new Thread
    System.out.print
}

```

The screenshot shows a Java code editor with the following code:

```
public static void main(String[] args) {
    System.out.println("Hello! I am " + Thread.currentThread().getName());
    MyRunnable rn = new MyRunnable();
    Thread th = new Thread
    System.out.print
}
```

A code completion dropdown is open at the line `Thread th = new Thread`. The dropdown lists several constructor options for the `Thread` class:

- ♦ `Thread()` (highlighted in blue)
- ♦ `Thread(Runnable r)`
- ♦ `Thread(String string)`
- ♦ `Thread(Runnable r, String string)`
- ♦ `Thread(ThreadGroup tg, Runnable r)`
- ♦ `Thread(ThreadGroup tg, String string)`
- ♦ `Thread(ThreadGroup tg, Runnable r, String string)`
- ♦ `Thread(ThreadGroup tg, Runnable r, String string, long l)`

Анонимные классы. Использование в аргументах. Пример

```
public static void main(String[] args) {
    Thread th = new Thread(new Runnable() {
        @Override
        public void run() {
            // этот код выполнится в другом потоке
            System.out.println("Hello, I am " + Thread.currentThread().getName());
        }
    });
}
```

Функциональный интерфейс

Функциональный интерфейс – это интерфейс, который содержит **один** абстрактный метод, то есть описание метода без тела.

Функциональный интерфейс может содержать любое количество методов по умолчанию (default) или статических методов.

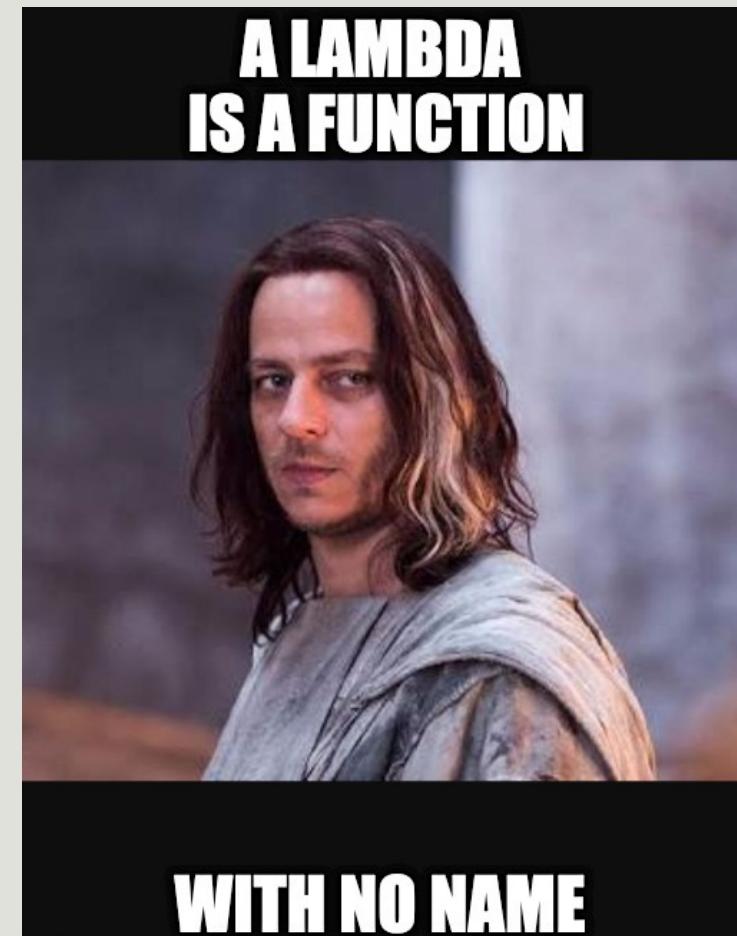
```
    /  
@FunctionalInterface  
public interface Runnable {  
    /**  
     * When an object implements  
     */
```



Что такое Lambda-выражения

Lambda-выражения – это компактный синтаксис, предназначенный для передачи кода в качестве параметра в другой код. По сути, это анонимный класс или метод.

Lambda-выражения появились в версии JDK 8 с целью усовершенствования языка Java.



Что такое Lambda-выражения

Синтаксис Lambda-выражений в Java: **(аргументы) -> (код метода)**

Например:

```
(арг1, арг2...) -> { тело }
```

```
(тип1 арг1, тип2 арг2...) -> { тело }
```

Использование Lambda-выражений.

Пример

```
public static void main(String[] args) {
    Thread th = new Thread(() -> {
        System.out.println("Hello, I am " + Thread.currentThread().getName());
    });
    th.start();
}
```

```
public static void main(String[] args) {
    new Thread(() -> System.out.println("Hello, I am " + Thread.currentThread().getName())).start();
}
```

<https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html#syntax>

Использование Lambda-выражений

- ❑ Lambda-выражения могут иметь от 0 и более входных параметров.
- ❑ Тип параметров можно указывать явно либо может быть получен из контекста. Например $(int\ a)$ можно записать и так (a)
- ❑ Если параметров нет, то нужно использовать пустые круглые скобки.
- ❑ Когда параметр один, если тип не указывается явно, скобки можно опустить. Пример: $a \rightarrow return\ a*a$
- ❑ Тело Lambda-выражения может содержать от 0 и более выражений.
- ❑ Если тело состоит из одного оператора, его можно не заключать в фигурные скобки, а возвращаемое значение можно указывать без ключевого слова `return`.

Использование Lambda-выражений.

Пример

```
@FunctionalInterface  
public interface SimpleCalc {  
    int calc(int x, int y);  
}
```

```
public static void main(String[] args) {  
    SimpleCalc sc = (x, y) -> x+y*2;  
    int rez = sc.calc( x: 4, y: 5 );  
    System.out.println("Result: " + rez);  
}
```



2. Java Stream API

Что такое Stream API

Stream API – это инструмент языка Java, который позволяет использовать функциональный стиль при работе с разными структурами данных и упрощать операции: фильтрации, сортировки и другие манипуляции.

Вся основная функциональность данного API сосредоточена в пакете **java.util.stream**.

Ключевым понятием в Stream API является поток данных.

Что такое Stream API

Stream API – это инструмент языка Java, который позволяет использовать функциональный стиль при работе с разными структурами данных и упрощать операции: фильтрации, сортировки и другие манипуляции.

Вся основная функциональность данного API сосредоточена в пакете **java.util.stream**.

Ключевым понятием в Stream API является поток данных.

Способы создания Stream



- Пустой stream: `Stream.empty()`
- Из List: `list.stream()`
- Из Map: `map.entrySet().stream()`
- Из массива: `Arrays.stream(array)`
- Из указанных элементов: `Stream.of("1", "2", "3")`
- Из указанных элементов-примитивов: `IntStream.of(9, 8, 7)`

Группы операторов

- **Промежуточные** («intermediate», ещё называют «lazy») – обрабатывают поступающие элементы и возвращают stream. Промежуточных операторов в цепочке обработки элементов может быть много.
- **Терминальные** («terminal») – обрабатывают элементы и завершают работу stream, так что терминальный оператор в цепочке может быть только один.

Промежуточные операторы

Метод	Что сделает
<code>filter</code>	отработает как фильтр, вернет значения, которые подходят под заданное условие
<code>sorted</code>	отсортирует элементы в естественном порядке; можно использовать Comparator
<code>limit</code>	лимитирует вывод по указанному количеству
<code>distinct</code>	найдет и уберет элементы, которые повторяются; вернет элементы без повторов
<code>peek</code>	выполнить действие над каждым элементом элементов, вернет stream с исходными элементами
<code>map</code>	выполнит действия над каждым элементом; вернет элементы с результатами обработки

Терминальные операторы

Метод	Что сделает
findFirst	вернет элемент, соответствующий условию, который стоит первым
findAny	вернет любой элемент, соответствующий условию
collect	соберет результаты обработки в коллекцию
count	посчитает и выведет, сколько элементов, соответствующих условию
min/max	найдет самый маленький (большой) элемент по переданному компаратору
forEach	применит функцию ко всем элементам, но порядок выполнения гарантировать не может
toArray	приведет значения стрима к массиву

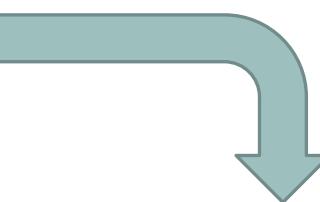
Пример использования Stream API

```
// вычислить сколько коротких слов в тексте (3 символа и меньше)
String text = "Студенты очень любят посещать лекции по Java и Android "
|     + "разработке. Всегда слушают очень внимательно и не спят.";
String[] words = text.split( regex: "\\\PL+" );
Stream stream = Arrays.stream( array:words );
long count = stream.filter(x -> x.toString().length() <= 3).count();

long count = stream.filter(new Predicate() {
    @Override
    public boolean test(Object x) {
        return x.toString().length() <= 3;
    }
}).count();
```

Пример использования Stream API

```
// вывести в консоль CAPS-ом все уникальные короткие слова из текста
String text = "Студенты очень любят посещать лекции по Java и Android "
    + "разработке. Всегда слушают очень внимательно и не спят.";
String[] words = text.split( regex: "\PL+" );
Stream stream = Arrays.stream( array:words );
stream.filter(x -> x.toString().length() <= 3)
    .map(t -> t.toString().toUpperCase())
    .distinct()
    .forEach(x -> System.out.println(x));
```



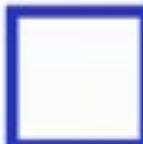
```
.map(new Function() {
    @Override
    public Object apply(Object t) {
        return t.toString().toUpperCase();
    }
})
```

Пример использования Stream API

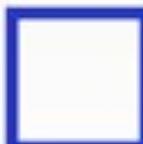


Stream

```
.of(120, 410, 85, 32, 314, 12)
```

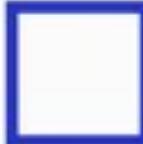


```
.filter(x -> x < 300)
```

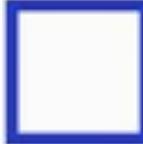


```
.map(x -> x + 11)
```

0



```
.limit(3)
```



```
.forEach(System.out::print)
```

Пример использования Stream API

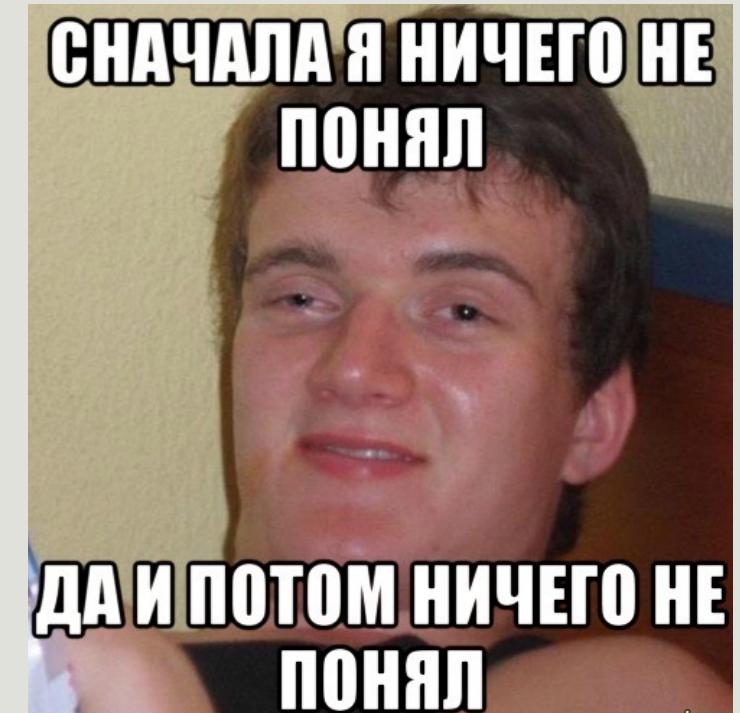
```
public class Person {  
    private String name;  
    private int age;  
    private String gender;  
    //...  
}  
  
// найти самого молодого мужчину из коллекции  
List<Person> persons = new ArrayList<>();  
persons.add(new Person( name:"Ivan Ivanov", age:25, gender:"male"));  
persons.add(new Person( name:"Petr Petrov", age:19, gender:"male"));  
persons.add(new Person( name:"Alex Petrov", age:21, gender:"male"));  
persons.add(new Person( name:"Lena Ivanova", age:17, gender:"female"));  
Person person = persons.stream() // получили поток из коллекции  
                    .filter(t -> t.getGender().equals("male"))  
                    .min((p1,p2) -> p1.getAge().compareTo(p2.getAge()))  
                    .get();  
System.out.println("Самый молодой мужчина из списка: " + person.getName());
```

Отличия коллекций от потоков

- Потоки не хранят элементы. Элементы, используемые в потоках, могут храниться в коллекции, либо при необходимости могут быть напрямую сгенерированы.
- Операции с потоками не изменяют источника данных. Операции с потоками лишь возвращают новый поток с результатами этих операций.
- Для потоков характерно отложенное выполнение. То есть выполнение всех операций с потоком происходит лишь тогда, когда выполняется терминальная операция и возвращается конкретный результат, а не новый поток.

Что почитать/посмотреть/порешать?

- Ричард Уорбертон, «Лямбда-выражения в Java 8»
- Рауль-Габриэль Урма, Марио Фуска, Алан Майкрофт, «Современный язык Java. Лямбда выражения, потоки и функциональное программирование»
- Stepic course: «Java. Functional Programming».
- Ok, Google!





3. Работа с сетью в Java

Общие понятия

Сетевые приложения на языке Java строятся с использованием технологии **Клиент/Сервер**. Соединение может быть установлено двумя способами:

- с помощью потоков (TCP)
- с помощью датаграмм (UDP)

Для работы с сетью в Java предусмотрена иерархия пакетов **java.net**



Сокеты

Сокет – это программная (логическая) конечная точка, устанавливающая двунаправленную коммуникацию между сервером и одной или несколькими клиентскими программами.

В памяти структура для сокета описывается двумя параметрами:

- **IP-адрес** – это по сути адрес компьютера в сети.
- **Порт** – это число (0-65535), которое должно быть уникально в рамках указанного компьютера. Только какое-то одно приложение (или служба) должно владеть этим портом в рамках операционной системы.

Сокет клиента

В Java клиентский сокет представлен классом **java.net.Socket**

Основные конструкторы класса **java.net.Socket**:

```
Socket(String имя_хоста, int порт) throws UnknownHostException, IOException  
Socket(InetAddress IP-адрес, int порт) throws UnknownHostException
```

Сокет сервера

В Java серверный сокет представлен классом **java.net.ServerSocket**

Основные конструкторы класса **java.net.ServerSocket**:

```
ServerSocket() throws IOException  
ServerSocket(int порт) throws IOException  
ServerSocket(int порт, int максимум_подключений) throws IOException  
ServerSocket(int порт, int максимум_подключений, InetAddress локальный_адрес) throws IOException
```

При объявлении **ServerSocket** не нужно указывать адрес соединения, потому что общение происходит на машине сервера.

Общая схема работы клиент-серверного приложения

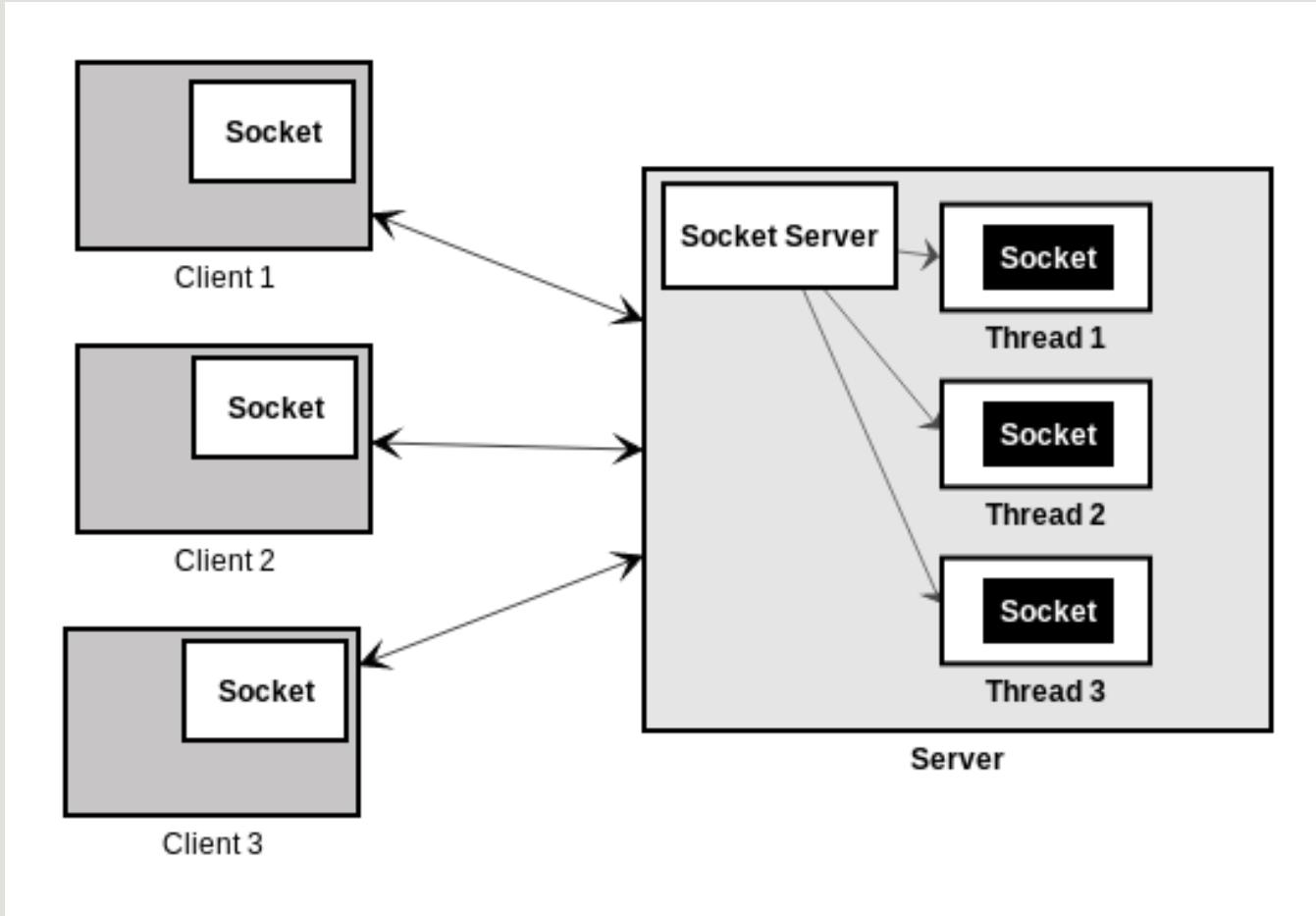
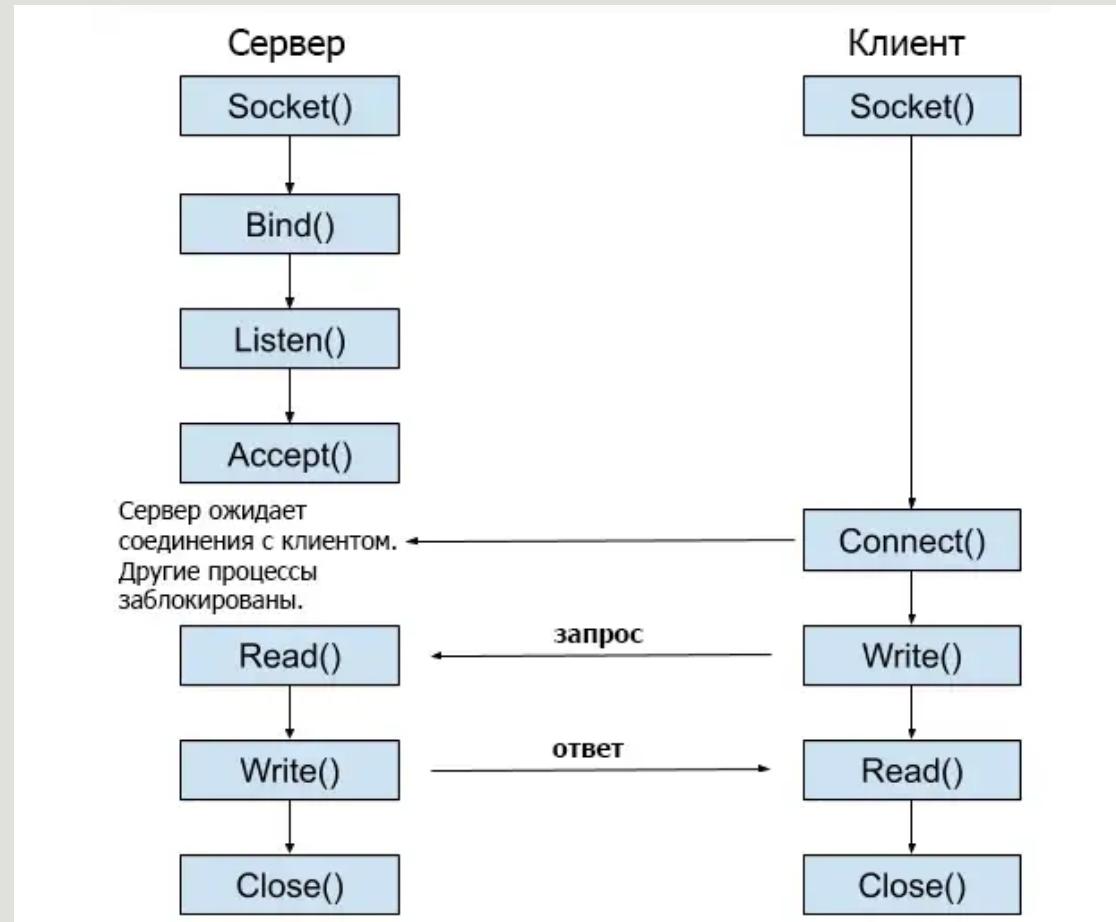


Схема работы соединения



Пример простого клиентского сокета

```
public class TCPClient {  
    public static void main (String[] args){  
        try {  
            Socket clientSocket = new Socket ("192.168.0.43", 30001); // подключение к серверу  
            OutputStream os = clientSocket.getOutputStream(); //Получение потока вывода из сокета  
            DataOutputStream dos = new DataOutputStream(os); // поток вывода для записи данных  
                                            // примитивных типов  
            dos.writeUTF("Привет от клиента " + clientSocket.getLocalSocketAddress()); // Запись  
                                            //строки в поток  
            clientSocket.close(); // закрытие сокета  
        }  
        catch (IOException e) {e.printStackTrace();}  
    }  
}
```

Пример простого серверного сокета

```
public class TCPServer {  
    public static void main (String[] args){  
        try {  
            ServerSocket serverSocket = new ServerSocket(30001); // создание серверного сокета  
            Socket socket = serverSocket.accept(); // Получение сокета, который подключен к  
                                            // конкретному клиенту  
            DataInputStream dis = new DataInputStream(socket.getInputStream());  
            String msgFromClient = in.readUTF(); // Чтение UTF-строк из потока ввода  
            System.out.println("Сообщение от клиента: " + msgFromClient);  
            socket.close();  
        } catch (SocketTimeoutException s | IOException e) {e.printStackTrace();}  
    }  
}
```

HTTP(S)



Протокол передачи гипертекста (**Hypertext Transfer Protocol - HTTP**) - это прикладной протокол для передачи гипертекстовых документов, таких как HTML. Он создан для связи между веб-браузерами и веб-серверами. Протокол следует классической клиент-серверной модели (клиент открывает соединение для создания запроса, а затем ждёт ответа).

Работа с HTTP(s). Пример GET-запроса

```
try {
    URL url = new URL(string:"https://www.mirea.ru/");
    URLConnection urlConnection = url.openConnection();
    HttpURLConnection connection = (HttpURLConnection) urlConnection;
    connection.setRequestMethod(method:"GET");
    InputStreamReader isr = new InputStreamReader(in:connection.getInputStream());
    BufferedReader in = new BufferedReader(reader:isr);
    StringBuilder urlString = new StringBuilder();
    String current;
    while((current = in.readLine()) != null) {
        urlString.append(str:current);
    }
    System.out.println(x:urlString);
} catch(IOException e){e.printStackTrace();}
```

Работа с HTTP(s)

```
<!DOCTYPE html><html lang="ru"><head>    <meta name="viewport" content="width=device-width, initial-scale=1">    <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">    <title>МИРЭА - Российский технологический университет</title>    <link rel="shortcut icon" type="image/x-icon" href="/favicon.ico" />    <link rel="apple-touch-icon" sizes="180x180" href="/apple-touch-icon.png">    <link rel="icon" type="image/png" sizes="32x32" href="/favicon-32x32.png">    <link rel="icon" type="image/png" sizes="16x16" href="/favicon-16x16.png">    <link rel="manifest" href="/site.webmanifest">    <link rel="mask-icon" href="/safari-pinned-tab.svg" color="#5bbad5">    <meta name="msapplication-TileColor" content="#da532c">    <meta name="msapplication-TileImage" content="/mstile-144x144.png">    <meta name="theme-color" content="#ffffff">    <link href="https://fonts.googleapis.com/css?family=PT+Sans:400,700&subset=latin,cyrillic' rel='stylesheet' type='text/css'>    <link href="https://fonts.googleapis.com/css2?family=Podkova&display=swap" rel="stylesheet">    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" /><meta name="robots" content="index, follow" /><meta name="keywords" content="PTY МИРЭА, официальный сайт РТУ МИРЭА, МИРЭА - Российский технологический университет, государственный вуз" /><meta name="description" content="МИРЭА - Российский технологический университет" /><script type="text/javascript" data-skip-moving="true">(function(w, d) { var v = w.frameCacheVars = {'CACHE_MODE': 'HTMLCACHE', 'storageBlocks': [], 'dynamicBlocks': {'MzahGm': 'f68273efdl6a', 'EKwMx3': 'f68273efdl6a', 'vDVIKy': 'dc72d5b9f457'}, 'AUTO_UPDATE': true, 'AUTO_UPDATE_TTL': '120', 'version': '2'}; var inv = false;if (v.AUTO_UPDATE === false){if (v.AUTO_UPDATE_TTL && v.AUTO_UPDATE_TTL > 0){var lm = Date.parse(d.lastModified);if (!isNaN(lm)){var td = new Date().getTime();if ((lm + v.AUTO_UPDATE_TTL * 1000) >= td){w.frameRequestStart = false;w.preventDefaultUpdate = true;return;}}else{w.frameRequestStart = false;w.preventDefaultUpdate = true;return;}}}var r = w.XMLHttpRequest ? new XMLHttpRequest() : (w.ActiveXObject ? new w.ActiveXObject("Microsoft.XMLHTTP") : null);if (!r) { return; }w.frameRequestStart = true;var m = v.CACHE_MODE; var l = w.location; var x = new Date().getTime();var q = "?bxrand=" + x + (l.search.length > 0 ? "&" + l.search.substring(1) : "");var u = l.protocol + "://" + l.host + l.pathname + q;r.open("GET", u, true);r.setRequestHeader("BX-ACTION-TYPE", "get_dynamic");r.setRequestHeader("X-Bitrix-Composite", "get_dynamic");r.setRequestHeader("BX-CACHE-MODE", m);r.setRequestHeader("BX-CACHE-BLOCKS", v.dynamicBlocks ? JSON.stringify(v.dynamicBlocks) : "");if (inv){r.setRequestHeader("BX-INVALIDATE-CACHE", "Y");}try { r.setRequestHeader("BX-REF", d.referrer || "");} catch(e) {}if (m === "APPCACHE"){r.setRequestHeader("BX-APPCACHE-PARAMS", JSON.stringify(v.PARAMS));r.setRequestHeader("BX-APPCACHE-URL", v.PAGE_URL ? v.PAGE_URL : "");}r.onreadystatechange = function() {if (r.readyState != 4) { return; }var a = r.getResponseHeader("BX-RAND");var b = w.BX && w.BX.frameCache ? w.BX.frameCache : false;if (a != x || !(r.status >= 200 && r.status < 300) || r.status === 304 || r.status === 1223 || r.status === 0)){var f = {error:true, reason:a!=x?"bad_rand":"bad_status", url:u,
```

ВОПРОСЫ?
