



УНИВЕРСИТЕТ ИТМО

Программирование. 2 семестр

Файлы (IO / NIO.2)

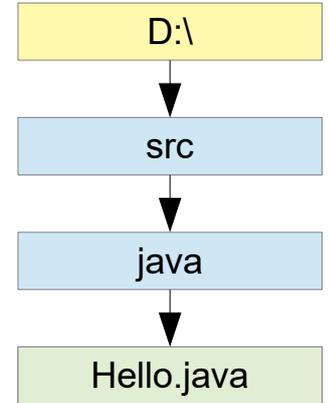


```
Compiled from "Hello.java" public class Hello minor version: 0 major
version: 52 flags: ACC_PUBLIC ACC_SUPER Constant pool: #1 = Methodref
#6.#15 // java/lang/Object; ldc #2 = String Hello world! //
java/lang/System.out: Ljava/io/PrintStream; #3 = String #18 // Hello world!
#4 = Methodref #19.#20 // java/io/PrintStream.println:(Ljava/lang/String;)V
#5 = Class #21 // Hello #6 = Class #22 // java/lang/Object #7 = Utf8
<init> #8 = Utf8 ()V #9 = Utf8 Code #10 = Utf8 LineNumberTable #11 =
Utf8 main #12 = Utf8 ([Ljava/lang/String;)V #13 = Utf8 SourceFile #14 =
Utf8 Hello.java #15 = NameAndType #7:#8 // <init>:()V #16 = Class #23
// java/lang/System.out: Ljava/io/PrintStream; ldc #24 = String Hello world!
#18 = Utf8 Hello world! #19 = Class #26 // java/io/PrintStream #20 =
NameAndType #27:#28 // println:(Ljava/lang/String;)V #21 = Utf8 Hello #22
= Utf8 java/lang/Object #23 = Utf8 java/lang/System #24 = Utf8 out #25
= Utf8 Ljava/io/PrintStream; #26 = Utf8 java/io/PrintStream #27 = Utf8
println #28 = Utf8 (Ljava/lang/String;)V { public Hello(): descriptor: ()V
flags: ACC_PUBLIC ACC_STATIC Code: stack=2, locals=1, args_size=1 0:
aload_0 1: invokespecial #1 // Method java/lang/Object.<init>:()V 4: return
LineNumberTable: line 1: 0 public static void main(java.lang.String[]);
descriptor: ([Ljava/lang/String;)V flags: ACC_PUBLIC, ACC_STATIC Code:
stack=2, locals=1, args_size=1 0: getstatic #2 // Field
java/lang/System.out: Ljava/io/PrintStream; 3: ldc #3 // String Hello world!
5: invokevirtual #4 // Method java/io/PrintStream.println:
(Ljava/lang/String;)V 8: return
SourceFile: "Hello.java"
LineNumberTable: line 3: 0 line 4: 8 }
```

Работа с файлами

- ✓ `java.nio.file.Path` — путь к файлу
 - абсолютный / относительный
 - пустой — текущая директория
- ✓ `java.nio.file.Paths`

```
Path p = Paths.get("java", "Hello.java");
```



java.io.File ↔ java.nio.file.Path

Path File.toPath()

File Path.toFile()

✓ `java.nio.file.FileSystem` — файловая система

✓ Методы:

```
Iterable<Path> getRootDirectories()
```

```
Path getPath(String first, String... more)
```

```
String getSeparator()
```

✓ `java.nio.file.FileSystems`

```
FileSystem getDefaultDefault()
```

☑ Класс Files

```
boolean Files.exists(Path)
```

```
boolean Files.notExists(Path)
```

```
boolean Files.isReadable(Path)
```

```
boolean Files.isWritable(Path)
```

```
boolean Files.isExecutable(Path)
```

Работа с файлами: create/delete

```
Path Files.createFile(Path)
```

```
Path Files.createTempFile(String prefix, String suffix)
```

```
void Files.delete(Path)
```

```
boolean Files.deleteIfExists(Path)
```

Работа с файлами: чтение и запись

```
byte[] Files.readAllBytes(Path)
```

```
List<String> Files.readAllLines(Path)
```

```
Files.write(Path, byte[])
```

```
Files.write(Path, Iterable<CharSequence>)
```

Работа с файлами: чтение и запись

```
FileInputStream / FileReader / FileOutputStream / FileWriter  
new BufferedXXX(new FileXXX(File/String))
```

java.io

```
Files.newInputStream(Path, OpenOptions...)  
Files.newOutputStream(Path, OpenOptions...)  
Files.newBufferedReader(Path, Charset, OpenOptions...)  
Files.newBufferedWriter(Path, Charset, OpenOptions...)
```

java.nio

```
String line;  
while ((line = reader.readLine()) != null) { }
```

чтение

```
String text;  
writer.write(s, 0, s.length());
```

запись



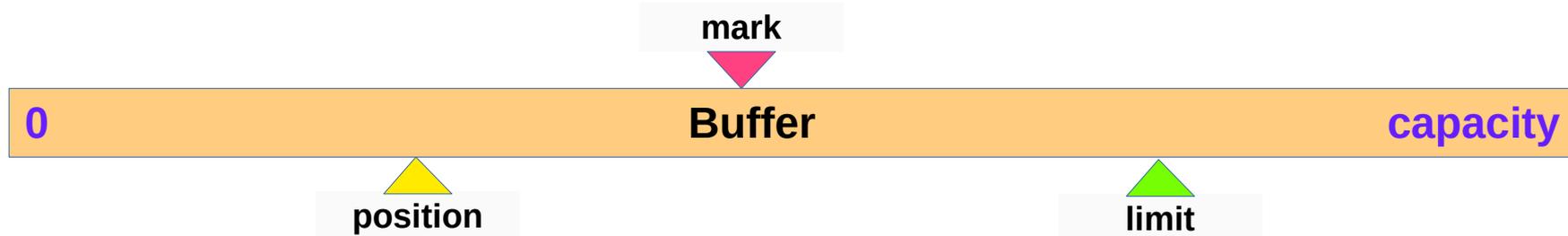
УНИВЕРСИТЕТ ИТМО

Программирование. 2 семестр

Ввод-вывод (NIO)

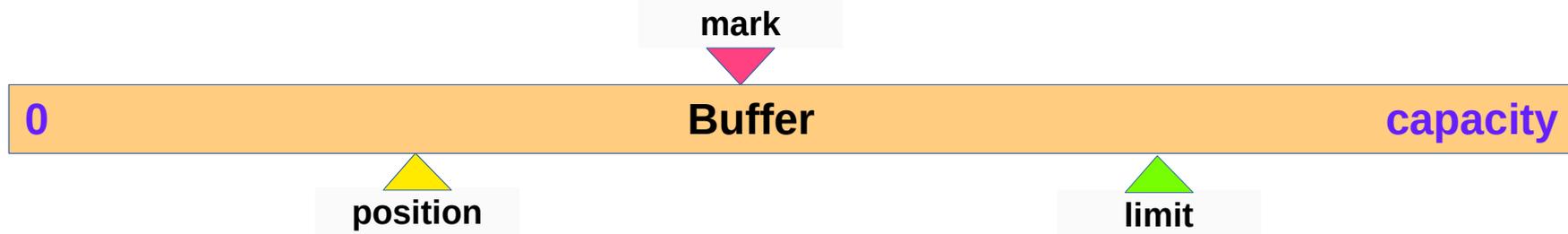


```
Compiled from "Hello.java" public class Hello minor version: 0 major
version: 52 flags: ACC_PUBLIC ACC_SUPER Constant pool: #1 = Methodref
#6.#15 // java/lang/Object;=Ljava/lang/Object;=Ljava/lang/Object;=
java/lang/System.out: Ljava/io/PrintStream; #3 = String #18 // Hello world!
#4 = Methodref #19.#20 // java/io/PrintStream.println:(Ljava/lang/String;)V
#5 = Class #21 // Hello #6 = Class #22 // java/lang/Object #7 = Utf8
<init> #8 = Utf8 ()V #9 = Utf8 Code #10 = Utf8 LineNumberTable #11 =
Utf8 main #12 = Utf8 ([Ljava/lang/String;)V #13 = Utf8 SourceFile #14 =
Utf8 Hello.java #15 = NameAndType #7:#8 // <init>:()V #16 = Class #23
// java/lang/System.out: Ljava/io/PrintStream; #17 = Utf8 out
#18 = Utf8 Hello world! #19 = Class #26 // java/io/PrintStream #20 =
NameAndType #27:#28 // println:(Ljava/lang/String;)V #21 = Utf8 Hello #22
= Utf8 java/lang/Object #23 = Utf8 java/lang/System #24 = Utf8 out #25
= Utf8 Ljava/io/PrintStream; #26 = Utf8 java/io/PrintStream #27 = Utf8
println #28 = Utf8 (Ljava/lang/String;)V { public Hello(): descriptor: ()V
flags: ACC_PUBLIC ACC_STATIC ACC_STATIC Code:
invokespecial #1 // java/lang/Object.<init>:()V 0: aload_0 1:
LineNumberTable: line 1: 0 public static void main(java.lang.String[]);
descriptor: ([Ljava/lang/String;)V flags: ACC_PUBLIC ACC_STATIC Code:
stack=2, locals=1, args_size=1 0: getstatic #2 // Field
java/lang/System.out: Ljava/io/PrintStream; 3: ldc #3 // String Hello world!
5: invokevirtual #4 // Method java/io/PrintStream.println:
(Ljava/lang/String;)V 8: return
SourceFile: "Hello.java"
```



✓ `java.nio.Buffer` —
контейнер для хранения
данных

- ✓ `capacity` - емкость (макс. размер)
- ✓ `limit` - сколько можно записать (не больше емкости) или прочитать (не больше, чем записано)
- ✓ `position` - текущая позиция



✓ Buffer — контейнер для хранения данных

✓ Создание буфера:

- `allocate(capacity)`
- `allocateDirect(capacity)`
- `wrap(array[])`

✓ Методы:

- `limit(lim)` и `position(pos)`
- `mark()` и `reset()` `mark <-> position`
- `clear()` - `limit = capacity`, `position = 0`
- `compact` - все недочитанное - в начало буфера
- `flip()` - `limit = position`, `position = 0`
- `rewind()` - `position = 0`

☑ методы get и put

- get - чтение из буфера
- put - запись в буфер

☑ Абсолютная индексация (явное указание индекса)

- позиция не меняется
- только одиночные операции

☑ Относительная индексация (по текущей позиции)

- позиция смещается после операции
- одиночные и групповые

Запись и чтение данных

✓ запись

`clear();`

```
while () { put(byte); }
```

✓ чтение

`flip();`

```
while(hasRemaining()) { get(); }
```



Запись и чтение данных

✓ запись

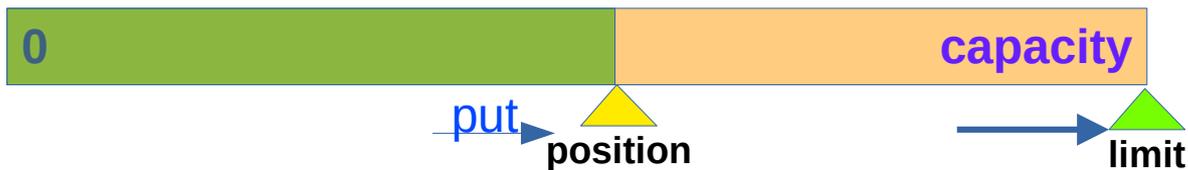
```
clear();
```

```
while () { put(byte); }
```

✓ чтение

```
flip();
```

```
while(hasRemaining()) { get(); }
```



Запись и чтение данных

✓ запись

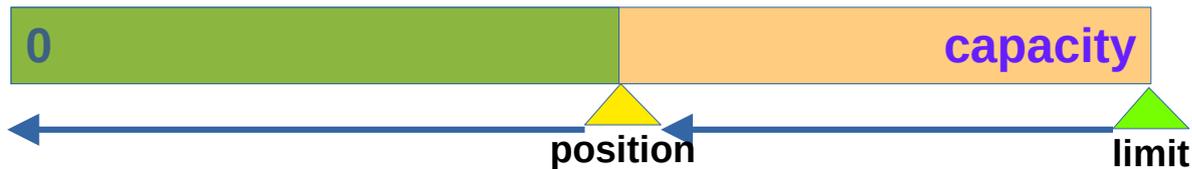
```
clear();
```

```
while () { put(byte); }
```

✓ чтение

```
flip();
```

```
while(hasRemaining()) { get(); }
```



Запись и чтение данных

✓ запись

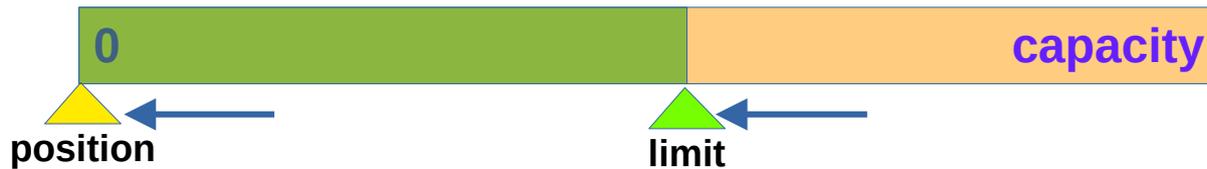
```
clear();
```

```
while () { put(byte); }
```

✓ чтение

```
flip();
```

```
while(hasRemaining()) { get(); }
```



Запись и чтение данных

✓ запись

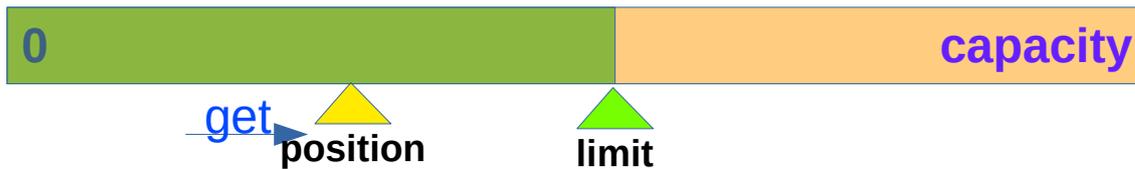
```
clear();
```

```
while () { put(byte); }
```

✓ **ЧТЕНИЕ**

```
flip();
```

```
while(hasRemaining()) { get(); }
```



- ✓ ByteBuffer
- ✓ CharBuffer
- ✓ IntBuffer, ShortBuffer, LongBuffer, FloatBuffer, DoubleBuffer

☑ Класс Charset

- МЕТОДЫ
- CharBuffer decode(ByteBuffer b)
- ByteBuffer encode(CharBuffer c)

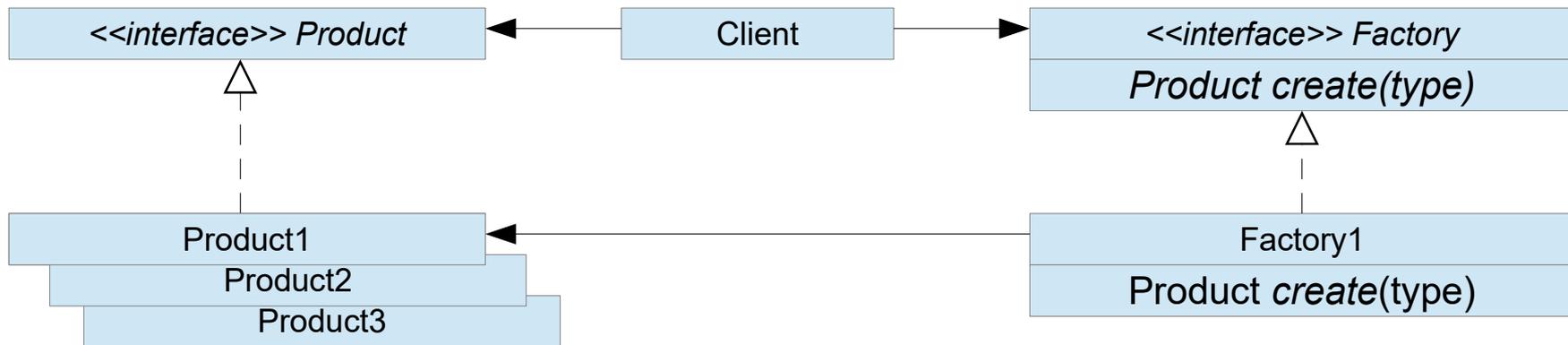
- ✓ `java.nio.channels`
- ✓ **Файловые каналы** и сетевые каналы
- ✓ **Отличие от потоков**
 - один канал для чтения и записи
 - поддержка неблокирующего ввода-вывода
 - поддержка асинхронного ввода-вывода
 - чтение и запись целого буфера

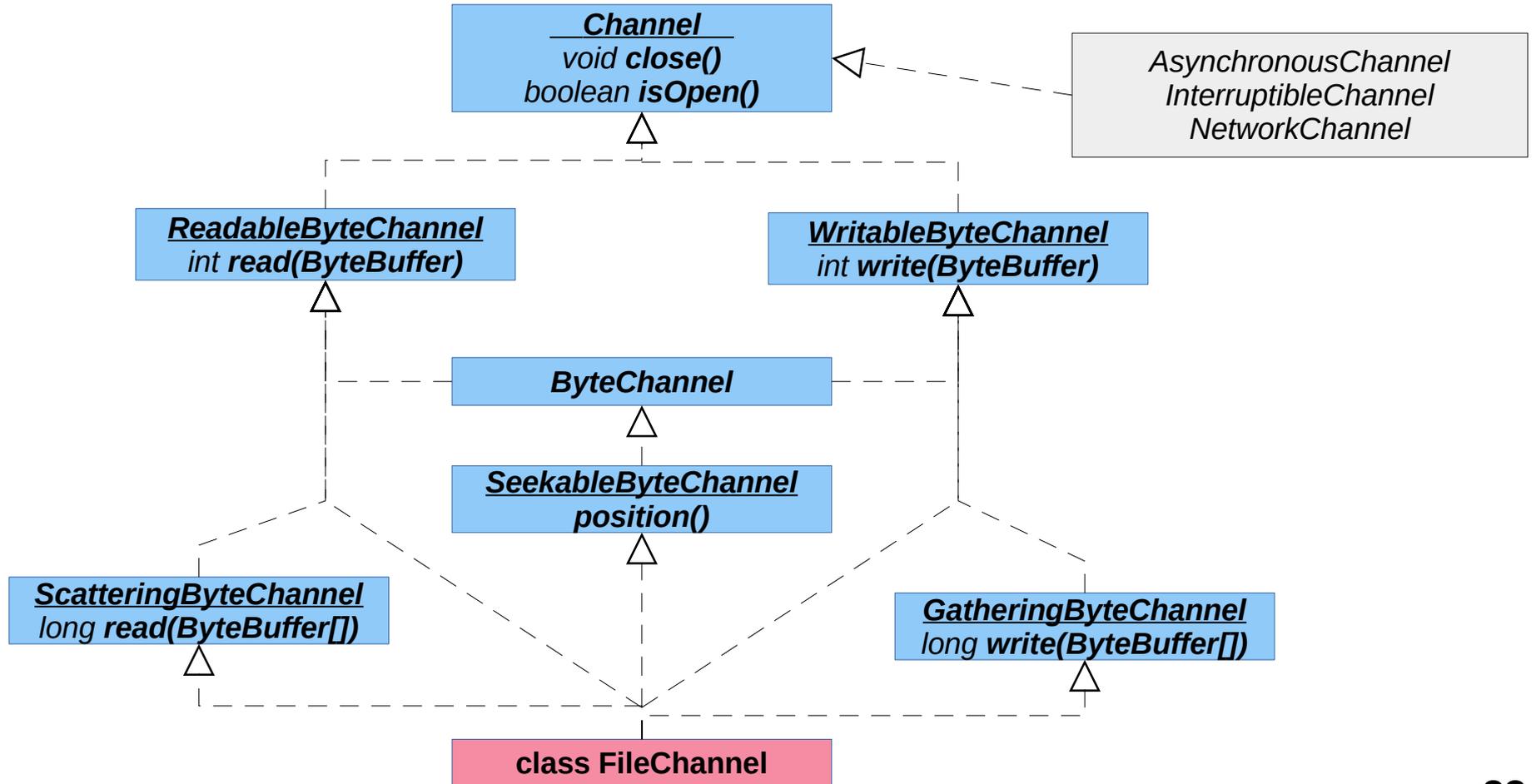
☑ Опять Теремок

- Блин с ветчиной и сыром
- Блин с куриной грудкой
- Блин с маслом
- Блин Фермерский

```
class PancakeFactory {  
    getCheeseHamPancake()  
    getPancake("CheeseHam")  
  
    getChickenPancake()  
    getButterPancake()  
    getFarmerPancake()  
}
```

- ✓ Создание объектов разных классов независимо от реализации





✓ фабричные методы:

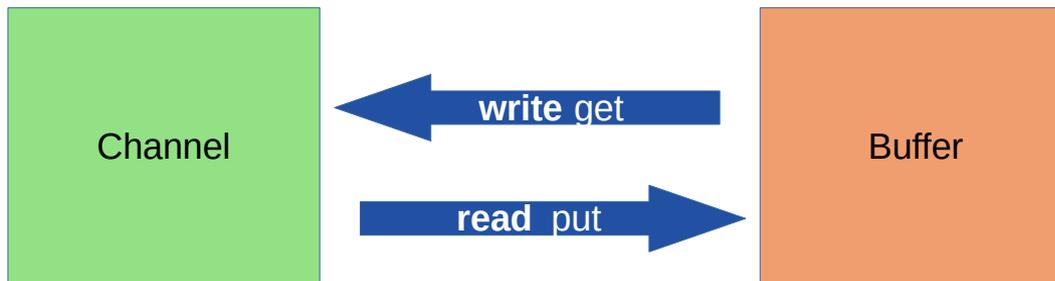
- `FileChannel.open()`
- `FileInputStream.getChannel()`

✓ `write(ByteBuffer b)`

- запись в канал из буфера

✓ `read(ByteBuffer b)`

- чтение из канала в буфер



☑ Чтение из файла

```
ByteBuffer buffer =  
ByteBuffer.allocate(1000);  
Path path = Paths.get("in.txt");  
FileChannel channel =  
    FileChannel.open(path);  
buffer.clear();  
int nBytes = channel.read(buffer);
```

☑ Запись в файл

```
Path path = Paths.get("out.txt");  
FileChannel channel =  
    FileChannel.open(path);  
buffer.flip();  
int nBytes = channel.write(buffer);
```

- ✓ Передача данных из канала в канал

```
transferFrom(ReadableByteChannel, long position, long count)
```

```
transferTo(long position, long count, WritableByteChannel)
```

- ✓ ScatteringByteChannel / GatheringByteChannel

```
ByteBuffer bufferArray = new ByteBuffer[3];
```

```
for (ByteBuffer buf : bufferArray) { buf.allocate(256); }
```

```
scatteringChannel.read(bufferArray);
```

```
gatheringChannel.write(bufferArray);
```

☑ FileChannel

- `map()` - получение `MappedByteBuffer`
- отображение файла в память

☑ `MappedByteBuffer`

- `boolean isLoaded()`
- `load()`
- `force()`

```
Compiled from "Hello.java" public class Hello minor version: 0 major
version: 52 flags: ACC_PUBLIC, ACC_SUPER Constant pool: #1 = Methodref
#6.#15 // java/lang/Object; #2 = Methodref #16.#17 //
java/lang/System.out: Ljava/io/PrintStream; #3 = String #18 // Hello world!
#4 = Methodref #19.#20 // java/io/PrintStream.println:(Ljava/lang/String;)V
#5 = Class #21 // Hello #6 = Class #22 // java/lang/Object #7 = Utf8
<init> #8 = Utf8 ()V #9 = Utf8 Code #10 = Utf8 LineNumberTable #11 =
Utf8 main #12 = Utf8 ([Ljava/lang/String;)V #13 = Utf8 SourceFile #14 =
Utf8 Hello.java #15 = NameAndType #7:#8 // <init>:()V #16 = Class #23
// java/lang/System.out: Ljava/io/PrintStream; #17 = Utf8
#18 = Utf8 Hello world! #19 = Class #26 // java/io/PrintStream #20 =
NameAndType #27:#28 // println:(Ljava/lang/String;)V #21 = Utf8 Hello #22
= Utf8 java/lang/Object #23 = Utf8 java/lang/System #24 = Utf8 out #25
= Utf8 Ljava/io/PrintStream; #26 = Utf8 java/io/PrintStream #27 = Utf8
println #28 = Utf8 (Ljava/lang/String;)V { public Hello(); descriptor: ()V
flags: ACC_PUBLIC, ACC_STATIC Code: line 1: 0 return
invokespecial #1 // method java/lang/Object.<init>:()V #4
LineNumberTable: line 1: 0 public static void main(java.lang.String[]);
descriptor: ([Ljava/lang/String;)V flags: ACC_PUBLIC, ACC_STATIC Code:
stack=2, locals=1, args_size=1 0: getstatic #2 // Field
java/lang/System.out: Ljava/io/PrintStream; 3: ldc #3 // String Hello world!
5: invokevirtual #4 // Method java/io/PrintStream.println:
(Ljava/lang/String;)V 8: return
SourceFile: "Hello.java"
```



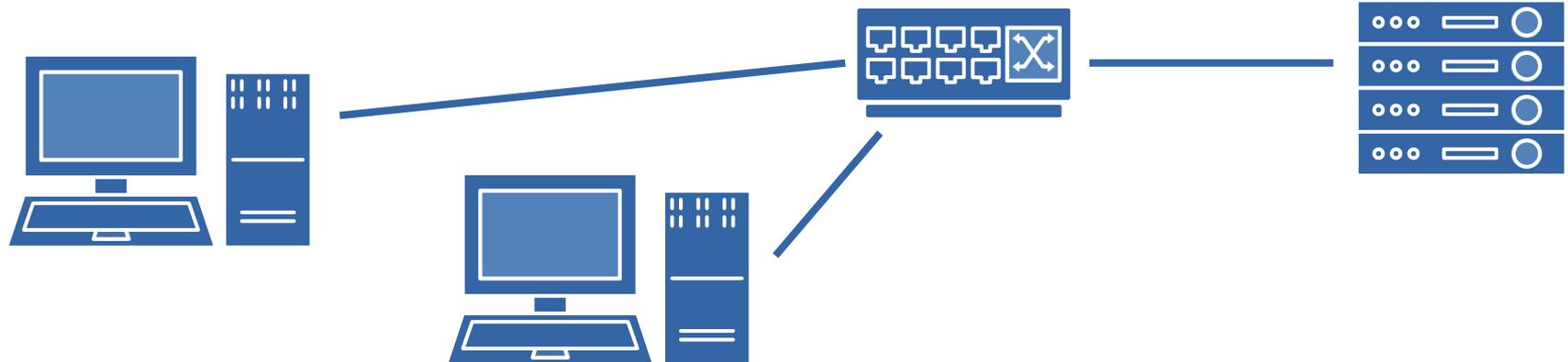
УНИВЕРСИТЕТ ИТМО

Программирование. 2 семестр

Сетевое взаимодействие



- ✓ *Вычислительная сеть* — система для передачи данных между узлами сети.
- ✓ *Хост* — компьютер, подключенный к сети и имеющий сетевой адрес.
- ✓ *Протокол* — набор правил, определяющих порядок действий и формат данных при сетевом обмене.



☑ *Клиент-серверная архитектура*

- Централизованное управление и обмен данными
- Сервер предоставляет сервисы в режиме ожидания запроса
- Клиент получает результат от сервера по запросу
- Надежность зависит от сервера — критический узел

☑ *Одноранговая архитектура (пиринговая)*

- Децентрализованное управление и обмен данными
- Все узлы (peers) равноправны, могут быть клиентом и сервером
- Нет критического узла

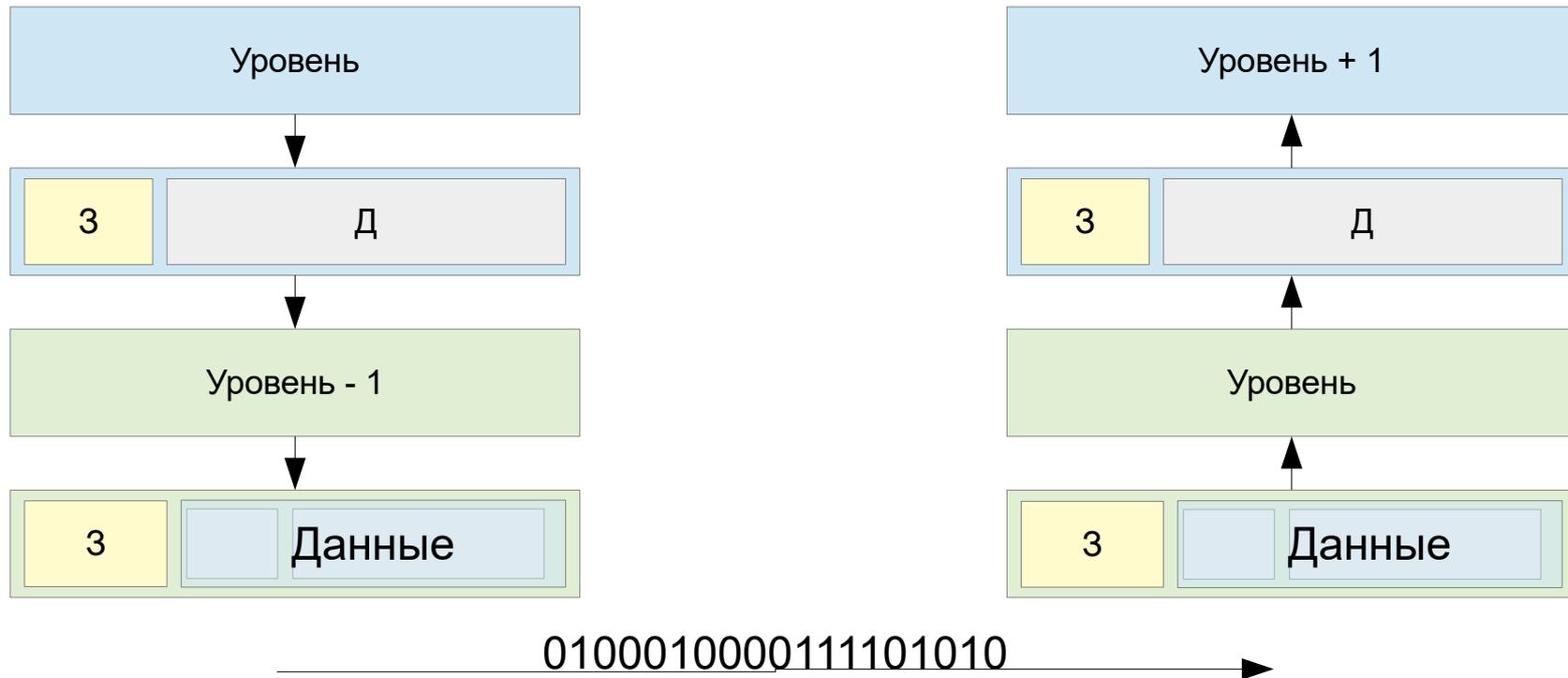
Модель передачи данных

☑ модель ISO/OSI

☑ стек TCP/IP

TCP/IP	OSI	Пример
Прикладной	Прикладной	HTTP
	Представительский	
	Сеансовый	
Транспортный	Транспортный	TCP, UDP
Сетевой	Сетевой	IP
Канальный	Канальный	Ethernet
	Физический	витая пара

☑ Пакет = заголовок + данные



☑ TCP

- устанавливается соединение
- подтверждение доставки
- **надежность** передачи данных

☑ UDP

- без установление соединения
- без подтверждения доставки
- **скорость** передачи данных

☑ Протокол прикладного уровня - HTTP

- ✓ Идентифицирует связь между роутером и хостом
- ✓ ID сети (префикс) + ID хоста (суффикс)
- ✓ IPv4 — 32 бита (194.85.160.55)
 - Класс А: префикс 8 бит (0...) + суффикс 24 бита
 - Класс В: префикс 16 бит (10...) + суффикс 16 бит
 - Класс С: префикс 24 бита (110...) + суффикс 8 бит
 - Маска подсети: 192.85.160.55
 - ◆ 192.168.0.5/255.255.255.240
 - ◆ 192.168.0.5/28
- ✓ IPv6 — 128 бит [FC05::4429:0:AC02]
- ✓ Loopback (localhost - 127.0.0.1 / [::1])

- ✓ DNS — Domain Name Service
- ✓ Преобразование между доменным именем и IP-адресом
- ✓ `www.google.com` ↔ `172.217.23.132`

- ✓ IP-адрес идентифицирует хост
- ✓ порт идентифицирует процесс (приложение)
- ✓ *Сокет* — интерфейс для обмена — адрес и порт

- ✓ Для обмена данными нужно знать:
 - протокол
 - IP-адрес и порт отправителя
 - IP-адрес и порт получателя

- ☑ Класс InetAddress
 - InetAddress Inet4Address Inet6Address
- ☑ Методы InetAddress (статические)
 - InetAddress getLocalHost()
 - InetAddress getByAddress(byte[] addr)
 - InetAddress getByName(String name) — обращение к DNS
- ☑ Нестатические методы
 - byte[] getAddress
 - String getHostName()

InetAddress

- ✓ `InetAddress(InetAddress addr, int port)`
- ✓ `InetAddress(int port)`
- ✓ `InetAddress(String hostname, int port)`

☑ Клиент

- Работает на **любом** хосте (сервер не знает, где именно)
- **Свободный** порт выбирается при отправлении запроса
- Посылает запрос серверу, ждет ответ

☑ Запрос

- Содержит данные и информацию о клиенте

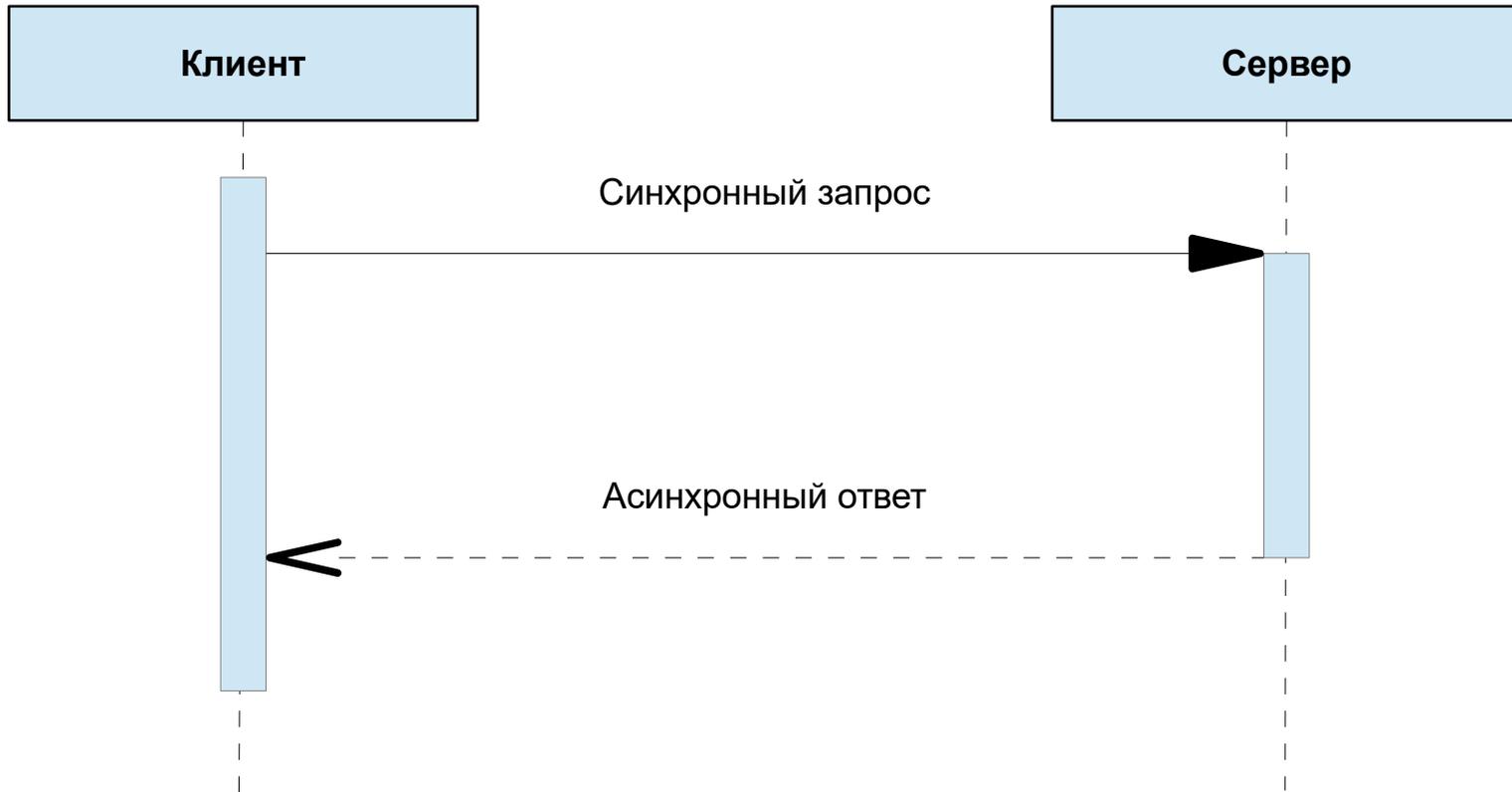
☑ Сервер

- Работает на **известном** хосте (известный IP-адрес)
- Прослушивает **известный** порт (зависит от сервиса)
- Ждет запрос от клиента, посылает ответ

☑ Ответ

- Содержит данные

Диаграмма последовательностей



- ☑ `java.net.DatagramPacket` — дейтаграмма (передаваемые данные + служебная информация)
 - Адрес буфера
 - Длина буфера
 - Адрес получателя (при отправлении)
- ☑ `java.net.DatagramSocket` — сокет для обмена
 - Порт для прослушивания (для получения)
 - Адрес и порт (для отправления)

Пример обмена по протоколу UDP

// клиент

```
byte arr[] = {0,1,2,3,4,5,6,7,8,9};
int len = arr.length;
DatagramSocket ds; DatagramPacket dp;
InetAddress host; int port;

ds = new DatagramSocket();

host = InetAddress.get...();
port = 6789;
dp = new DatagramPacket(arr, len, host, port);
ds.send(dp);

dp = new DatagramPacket(arr, len);
ds.receive(dp);

for (byte j : arr) {
    System.out.println(j);
}
```

// сервер

```
byte arr[] = new byte[10];
int len = arr.length;
DatagramSocket ds; DatagramPacket dp;
InetAddress host; int port = 6789;

ds = new DatagramSocket(port);

dp = new DatagramPacket(arr, len);
ds.receive(dp);

for (int j = 0; j < len; j++) {
    arr[j] *= 2;
}

host = dp.getAddress();
port = dp.getPort();
dp = new DatagramPacket(arr, len, host, port);
ds.send(dp);
```

- ☑ `java.net.Socket` — сокет для обмена (клиент и сервер)
 - `new Socket` (адрес + порт — для отправления)
 - `Socket ServerSocket.аccept()` - для получения
- ☑ `java.net.ServerSocket` — фабрика сокетов
 - `new ServerSocket(порт)` — на стороне сервера
- ☑ обмен данными через потоки ввода-вывода
 - `Socket.getInputStream()`
 - `Socket.getOutputStream()`

Пример обмена по протоколу TCP

// клиент

```
byte arr[] = {0,1,2,3,4,5,6,7,8,9};
int len = arr.length;
Socket sock;
OutputStream os; InputStream is;
InetAddress host; int port;

port = 6789;
sock = new Socket(host,port);

os = sock.getOutputStream();
os.write(arr);

is = sock.getInputStream();
is.read(arr);

for (byte j : arr) {
    System.out.println(j);
}
```

// сервер

```
byte arr[] = new byte[10];
int len = arr.length;
Socket sock; ServerSocket serv;
OutputStream os; InputStream is;
InetAddress host; int port = 6789;

serv = new ServerSocket(port);
sock = serv.accept();

is = sock.getInputStream();
is.read(arr);

for (int j = 0; j < len; j++) {
    arr[j] *= 2;
}

os = sock.getOutputStream();
os.write(arr);
```

☑ Протокол UDP — DatagramChannel

- DatagramChannel open()
- bind(SocketAddress local) // сервер
- SocketAddress receive(ByteBuffer)
- int send(ByteBuffer, SocketAddress)

- connect(SocketAddress remote) // клиент
- int read(ByteBuffer)
- int write(ByteBuffer)

Пример обмена по протоколу UDP (NIO)

// клиент

```
byte arr[] = {0,1,2,3,4,5,6,7,8,9};
int len = b.length;
DatagramChannel dc; ByteBuffer buf;
InetAddress host; int port;
SocketAddress addr;

addr = new InetSocketAddress(host,port);
dc = DatagramChannel.open();

buf = ByteBuffer.wrap(arr);
dc.send(buf, addr);

buf.clear();
addr = dc.receive(buf);

for (byte j : arr) {
    System.out.println(j);
}
```

// сервер

```
byte arr[] = new byte[10];
int len = arr.length;
DatagramChannel dc; ByteBuffer buf;
InetAddress host; int port = 6789;
SocketAddress addr;

addr = new InetSocketAddress(port);
dc = DatagramChannel.open();
dc.bind(addr);

buf = ByteBuffer.wrap(arr);
addr = dc.receive(buf);

for (int j = 0; j < len; j++) {
    arr[j] *= 2;
}

buf.flip();
dc.send(buf, addr);
```

☑ Протокол TCP

- ServerSocketChannel
 - ◆ ServerSocketChannel open()
 - ◆ bind(SocketAddress local)
 - ◆ SocketChannel accept() // сервер
- SocketChannel
 - ◆ SocketChannel connect(SocketAddress remote) / клиент
 - ◆ write(ByteBuffer)
 - ◆ read(ByteBuffer)

Пример обмена по протоколу TCP (NIO)

// клиент

```
byte arr[] = {0,1,2,3,4,5,6,7,8,9};
int len = arr.length;
InetAddress host; int port;
SocketAddress addr; SocketChannel sock;

port = 6789;
addr = new InetSocketAddress(host,port);
sock = SocketChannel.open();
sock.connect(addr);

buf = ByteBuffer.wrap(arr);
sock.write(buf);

buf.clear();
sock.read(buf);

for (byte j : arr) {
    System.out.println(j);
}
```

// сервер

```
byte arr[] = new byte[10];
int len = arr.length;
InetAddress host; int port = 6789;
SocketAddress addr; SocketChannel sock;
ServerSocketChannel serv;

serv = ServerSocketChannel.open();
serv.bind(port);
sock = serv.accept();

buf = ByteBuffer.wrap(arr);
sock.read(buf);

for (int j = 0; j < len; j++) {
    arr[j] *= 2;
}

buf.flip();
sock.write(buf);
```

☑ Блокирующий режим

- Можно ли выполнить операцию = попытаться выполнить
 - ◆ `setSoTimeout(long milliseconds)`
 - ◆ `Socket` / `ServerSocket` / `DatagramSocket`

☑ Неблокирующий режим

- Проверка возможности - отдельно от самой операции

☑ `SocketChannel.configureBlocking(false)`

- Методы `read` и `write` возвращают `int` — количество прочитанных байт, или `-1`, если данных больше нет.

☑ `ServerSocketChannel.configureBlocking(false)`

- Метод `accept()` возвращает `SocketChannel` или `null`, если соединение не установлено

☑ abstract class Selector

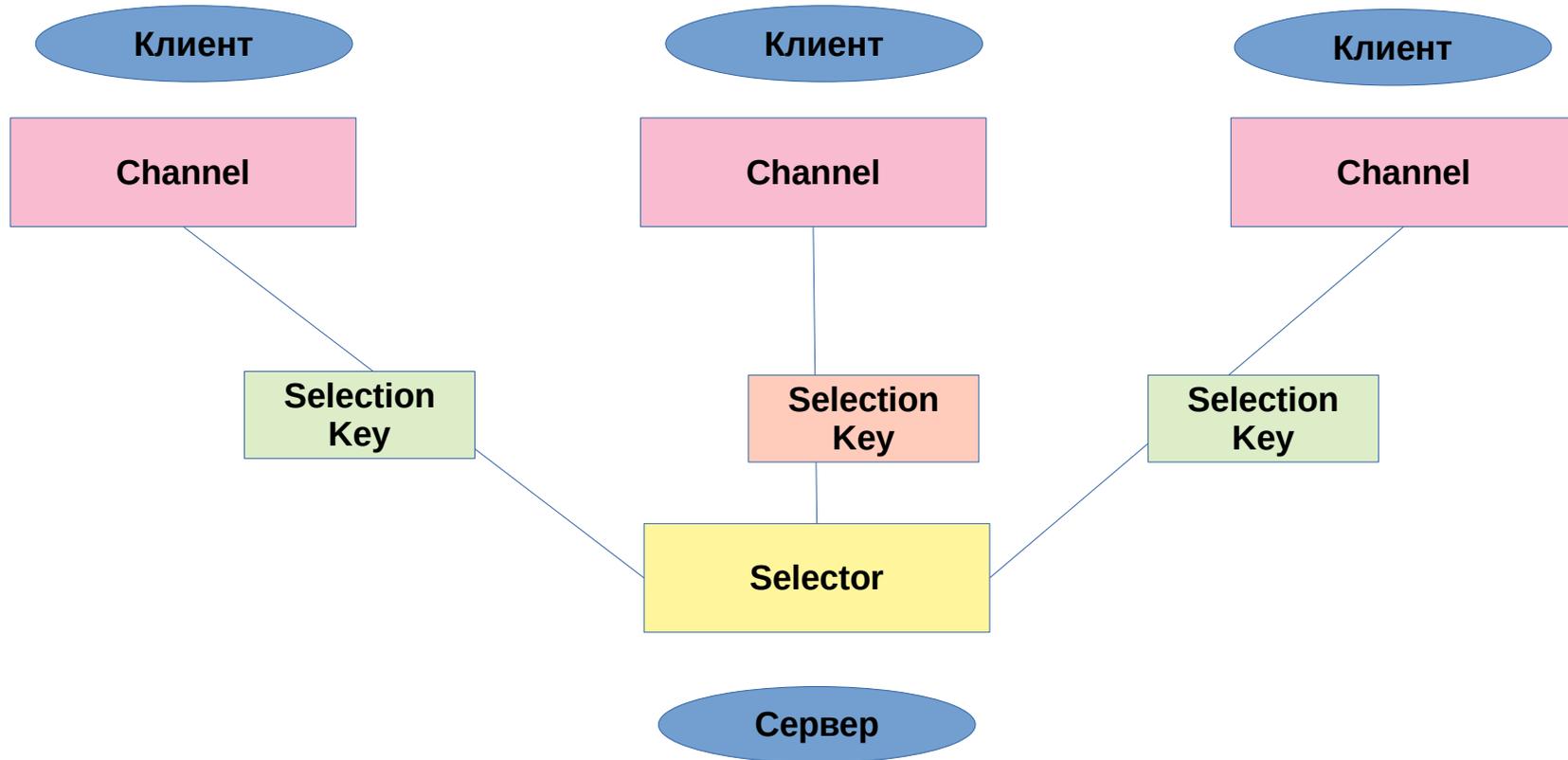
- open()
- select()
- Set<SelectionKey> keys() // key set
- Set<SelectionKey> selectionKeys() // selected key set
- // cancelled key set

☑ abstract class SelectionKey

- interestOps(), readyOps()
- OP_CONNECT, OP_ACCEPT, OP_READ, OP_WRITE
- isConnectable(), isAcceptable(), isReadable(), isWritable()
- channel(), selector()
- attach(Object), Object attachment()
- cancel()

☑ abstract class SelectableChannel

- SelectonKey `register(Selector s, int Ops, Object attachment)`
 - ◆ SocketChannel
 - ◆ ServerSocketChannel
 - ◆ DatagramChannel



```
Selector selector = Selector.open();
ServerSocketChannel server = ServerSocketChannel.open();
server.configureBlocking(false);
server.register(selector, SelectionKey.OP_ACCEPT);
while(true) {
    selector.select();
    Set<SelectionKey> keys = selector.selectedKeys();
    for (var iter = keys.iterator(); iter.hasNext(); ) {
        SelectionKey key = iter.next(); iter.remove();
        if (key.isValid()) {
            if (key.isAcceptable()) { doAccept(); }
            if (key.isReadable()) { doRead(); }
            if (key.isWritable()) { doWrite(); }
        }
    }
}
selector.close();
```

accept, read, write

```
doAccept() {  
    var ssc = (ServerSocketChannel) key.channel();  
    var sc = ssc.accept();  
    key.attach(clientData);  
    sc.configureBlocking(false);  
    sc.register(key.selector(), OP_READ);  
}
```

```
doRead() {  
    var sc = (SocketChannel) key.channel();  
    var data = (ClientData) key.attachment();  
    sc.read(data.buffer);  
    sc.register(key.selector(), OP_WRITE);  
}
```

```
doWrite() {  
    var sc = (SocketChannel) key.channel();  
    var data = (ClientData) key.attachment();  
    sc.write(data.buffer);  
}
```

- ☑ URI — Unified Resource Identifier
 - URL — Unified Resource Locator
 - URN — Unified Resource Name

```
URL url = new URL("http://www.google.com");  
  
InputStream is = url.openStream();  
is.read();  
is.close();  
  
Object o = url.getContent();
```

```
URL url = new URL("http://www.google.com");
URLConnection uc = url.openConnection();
uc.connect();

InputStream is = uc.getInputStream();
// is.read();

uc.setDoOutput(true);
OutputStream os = uc.getOutputStream();
// os.write();

is.close();
os.close();
uc.close()
```



УНИВЕРСИТЕТ ИТМО

Программирование. 2 семестр

Функциональное программирование

ITMO More than a
UNIVERSITY

- ✓ Функции высшего порядка
- ✓ Нет побочных эффектов
- ✓ Нет состояния
- ✓ Ленивые вычисления
- ✓ Достоинства:
 - Автоматическая многопоточность
 - Более надежный код — простота тестирования
 - Оптимизация

- ☑ Итерация → Рекурсия
- ☑ Проблема — ограничение стека
 - Вызов функции — параметры и адрес возврата — в стек
 - Во время работы функции локальные переменные в стеке
 - Возврат — очистка стека и переход по адресу возврата
- ☑ Решение — хвостовая рекурсия
 - Рекурсивный вызов функции — последняя команда
 - Вместо повторных рекурсивных вызовов — замена параметров и возврат к началу (фактически - итерация)

☑ Итерация

```
public int factor(int n) {  
    int result = 1;  
    int i = 1  
    while (i <= n) {  
        result *= i;  
        i += 1;  
    }  
    return result;  
}
```

☑ Итерация

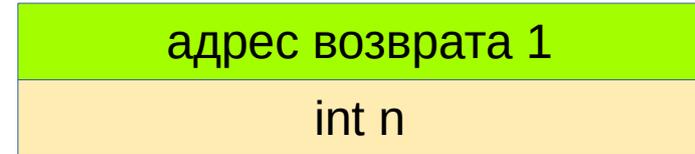
```
public int factor(int n) {  
    int result = 1;  
    int i = 1;  
    while (i <= n) {  
        result *= i;  
        i += 1;  
    }  
    return result;  
}
```

☑ Рекурсия

```
public int factor(int n) {  
    if (n <= 1) {  
        return 1;  
    } else {  
        return factor(n-1) * n;  
    }  
}
```

- проще код
- проблема стека вызовов

- ✓ При вызове метода - в стек помещаются параметры и адрес возврата
- ✓ При работе метода - в стек помещаются локальные переменные
- ✓ Перед возвратом - очистка локальных переменных
- ✓ Во время возврата - очистка от параметров и возврат



```
public int factor(int n) {  
    if (n <= 1) {  
        return 1;  
    } else {  
        return factor(n-1) * n;  
    }  
}
```

- ✓ При вызове метода - в стек помещаются параметры и адрес возврата
- ✓ При работе метода - в стек помещаются локальные переменные
- ✓ Перед возвратом - очистка локальных переменных
- ✓ Во время возврата - очистка от параметров и возврат

адрес возврата 1
int n
адрес возврата 2
int n

```
public int factor(int n) {  
    if (n <= 1) {  
        return 1;  
    } else {  
        return factor(n-1) * n;  
    }  
}
```

- ✓ При вызове метода - в стек помещаются параметры и адрес возврата
- ✓ При работе метода - в стек помещаются локальные переменные
- ✓ Перед возвратом - очистка локальных переменных
- ✓ Во время возврата - очистка от параметров и возврат

адрес возврата 1
int n
адрес возврата 2
int n
адрес возврата 3
int n

```
public int factor(int n) {  
    if (n <= 1) {  
        return 1;  
    } else {  
        return factor(n-1) * n;  
    }  
}
```

- ✓ При вызове метода - в стек помещаются параметры и адрес возврата
- ✓ При работе метода - в стек помещаются локальные переменные
- ✓ Перед возвратом - очистка локальных переменных
- ✓ Во время возврата - очистка от параметров и возврат

адрес возврата 1
int n
адрес возврата 2
int n

```
public int factor(int n) {  
    if (n <= 1) {  
        return 1;  
    } else {  
        return factor(n-1) * n;  
    }  
}
```

Факториал (хвостовая рекурсия)

✓ Хвостовая рекурсия

```
public int factor(int n, int f) {  
    if (n <= 1) {  
        return f;  
    } else {  
        return factor(n-1, f * n);  
    }  
}
```

✓ Рекурсия

```
public int factor(int n) {  
    if (n <= 1) {  
        return 1;  
    } else {  
        return factor(n-1) * n;  
    }  
}
```

```
public int factor(int n) {  
    return factor(n, 1);  
}
```

Факториал (хвостовая рекурсия)

✓ Хвостовая рекурсия

```
public int factor(int n, int f) {  
    if (n <= 1) {  
        return f;  
    } else {  
        return factor(n-1, f * n);  
    }  
}
```

```
public int factor(int n) {  
    return factor(n, 1);  
}
```

адрес возврата 1

int n

int f

Факториал (хвостовая рекурсия)

✓ Хвостовая рекурсия

```
public int factor(int n, int f) {  
    if (n <= 1) {  
        return f;  
    } else {  
        return factor(n-1, f * n);  
    }  
}
```

```
public int factor(int n) {  
    return factor(n, 1);  
}
```

адрес возврата 1
адрес возврата 2
int n
int f

☑ Хвостовая рекурсия

```
public int factor(int n, int f) {  
    if (n <= 1) {  
        return f;  
    } else {  
        return factor(n-1, f * n);  
    }  
}
```

```
public int factor(int n) {  
    return factor(n, 1);  
}
```

адрес возврата 1
int n
int f

- ☑ Алонзо Чёрч (Alonzo Church)
- ☑ $\hat{a}.a+1 \rightarrow \lambda a.a+1 \rightarrow \Lambda a.a+1 \rightarrow \lambda a.a+1$
 - \hat{a} — аргумент выражения $a+1$
- ☑ Переменная: x
- ☑ Операции:
 - Абстракция: $\lambda x.f$ (связывание x с функцией f)
 - Аппликация (применение): $f g$ (применение f к аргументу g)

✓ $inc(x) = x + 1$ $inc(3)$

✓ $f(x) = x + 1$ $f(3)$

✓ $(x) \rightarrow x + 1$ $((x) \rightarrow x + 1) (3)$

✓ $\lambda x. x + 1$

✓ Свободные и связанные переменные

✓ $\lambda x. x + y$

связанная

✓ $(x) \rightarrow x + y$

свободная

- ☑ Передача функции с целью ее дальнейшего вызова
 - реализация действия разными способами, выбираемыми во время исполнения
 - реализация асинхронной реакции на события
- ☑ Варианты реализации
 - указатели на функцию (C, C++)
 - делегаты (C#)
 - объект интерфейса с методом / анонимный класс (Java < 8)
 - λ-выражения (Java 8+)

Пример - список студентов

```
class Student {  
  
    public String getName() { ... }  
    public double getAge() { ... }  
    public double getAvgMark() { ... }  
    public String getGroup() { ... }  
    public String getEmail() { ... }  
  
    static List<Student> students;  
  
    public static void printAll() {  
        for (Student st : students) {  
            System.out.println(st.getName());  
        }  
    }  
}
```

```
class Student {
    ...

    public static void printAll() { ... }

    public static void printExcellentFromGroup(String g) {
        for (Student st : students) {
            if ((st.getGroup().equals(g) && (st.getAvgMark() > 4.75)) {
                System.out.println(st.getName());
            }
        }
    }
}
```

```
class Student {
    ...

    public static void printAll() { ... }

    public static void printExcellentFromGroup(String g) {...}

    public static void printExcellentAndYoung() {
        for (Student st : students) {
            if ((st.getAvgMark() > 4.75) && (st.getAge() < 20)) {
                System.out.println(st.getName());
            }
        }
    }
}
```

Список избранных :)

```
interface Checker {
    abstract public boolean test(Student st);
}

class Student {
    ...
    public void printSelected(Checker ch) {
        for (Student st : students) {
            if (ch.test(st)) {
                System.out.println(st.getName());
            }
        }
    }
}
```

```
interface Checker {
    abstract public boolean test(Student st);
}
class Student {
    ...
    public void printSelected(Checker ch) {...}
}

class ExcellentAndYoungChecker implements Checker {
    public boolean test(Student st) {
        return (st.getAge() < 20) && (st.getAvgMark() > 4.75));
    }
}
Student.printSelected(new ExcellentAndYoungChecker());
```

```
interface Checker {
    abstract public boolean test(Student st);
}
class Student {
    ...
    public void printSelected(Checker ch) {...}
}

Student.print(new Checker() {
    public boolean test(Student st) {
        return (st.getAge() < 20) && (st.getAvgMark() > 4.75));
    });
});
```

```
@FunctionalInterface
interface Checker {
    abstract public boolean test(Student st);
}
class Student {
    ...
    public void print(Checker ch) {...}
}

Student.print(new Checker() {
    public boolean test(Student st) {
        return (st.getAge() < 20) && (st.getAvgMark() > 4.75));
    });
```

```
@FunctionalInterface
interface Checker {
    abstract public boolean test(Student st);
}
class Student {
    ...
    public void print(Checker ch) {...}
}

Student.print(new Checker() {
    public boolean test(Student st) {
        return (st.getAge() < 20) && (st.getAvgMark() > 4.75);
    });
});
```

```
(Student st) -> (st.getAge() < 20) &&
(st.getAvgMark() > 4.75)
```

```
@FunctionalInterface
interface Checker {
    abstract public boolean test(Student st);
}
class Student {
    ...
    public void print(Checker ch) {...}
}

Student.print(
    (Student st) -> (st.getAge() < 20) && (st.getAvgMark() > 4.75))
);
```

```
@FunctionalInterface
interface Checker {
    abstract public boolean test(Student st);
}
class Student {
    ...
    public void print(Checker ch) {...}
}

Student.print(
    st -> (st.getAge() < 20) && (st.getAvgMark() > 4.75))
);

Student.print(st -> st.getGroup() == 3145);
```

```
import java.util.function.*;
interface Predicate<T> {
    abstract public boolean test(T t);
}
class Student {
    ...
    public void print(Predicate<Student> ch) {...}
}

Student.print(
    st -> (st.getAge() < 20) && (st.getAvgMark() > 4.75))
);

Student.print(st -> st.getGroup() == 3145);
```

```
import java.util.function.*;

class Student {
    ...
    public void print(Predicate<Student> ch) {
        for (Student st : students) {
            if (ch.test(st)) {
                System.out.println(st.getName());
            }
        }
    }
}

Student.print(st -> st.getGroup().equals("3145"));
```

```
import java.util.function.*;

class Student {
    ...
    public void handle(Predicate<Student> p,
                      Consumer<Student> c) {
        for (Student st : students) {
            if (p.test(st)) { c.accept(st); }
        }
    }
}

Student.handle(st -> st.getGroup().equals("3145"),
               st -> System.out.println(st));
```

+ Iterable - универсальный обработчик

```
import java.util.function.*;

class Student {
    ...
    public <X> void handle(Iterable<X> i,
                          Predicate<X> p,
                          Consumer<X> c) {
        for (X e : i) {
            if (p.test(e)) { c.accept(e); }
        }
    }
}

Student.handle(Student.students,
               s -> s.getGroup().equals("3145"),
               s -> System.out.println(s));
```

Ссылка на метод (Method Reference)

```
import java.util.function.*;

class Student {
    ...
    public <X> void handle(Iterable<X> i, Predicate<X> p, Consumer<X> c);
    public boolean checkGroup(String g) {
        return this.getGroup().equals(g);
    }
}

Student.handle(Student.students,
                s -> s.checkGroup("3145"),
                s -> System.out.println(s));
```

Ссылка на метод (Method Reference)

```
import java.util.function.*;

class Student {
    ...
    public <X> void handle(Iterable<X> i, Predicate<X> p, Consumer<X> c);
    public boolean checkGroup(String g) {
        return this.getGroup().equals(g);
    }
}

Student.handle(Student.students,
                s -> s.checkGroup("3145"),
                System.out::println);
```

- ☑ λ -выражение — блок кода для объявления анонимной функции.
- ☑ λ -выражение имеет целевой тип функционального интерфейса
- ☑ Функциональный интерфейс - только один абстрактный метод
 - не считая default и методов Object
- ☑ λ -выражение можно присвоить переменной
- ☑ λ -выражение можно передать методу
- ☑ λ -выражение можно вернуть из метода

параметр -> выражение

(параметры) -> { инструкции; }

(int x, int y) -> x + y

(x, y) -> x * y

() -> 42

(String s) -> System.out.println(s)

x -> x / 2

c -> { int s = c.size(); c.clear(); return s; }

- ☑ Область видимости λ -выражения = область видимости окружающего блока
- ☑ В λ -выражении можно использовать только эффективно финальные переменные из окружающего его блока
- ☑ λ -выражение захватывает значения переменных из окружающего блока.
- ☑ λ -выражение + значения захваченных переменных = замыкание (closure)

```
public static void repeatMessage(int count, String s) {  
    Runnable r = () -> {  
        for (int i = 0; i < count; i++) { println(s); }  
    }  
    new Thread(r).start();  
}
```

✓ `java.lang.Runnable`

```
void run();
```

```
new Thread(() -> { ... }).start();
```

✓ `java.util.Comparator<T>`

```
int compare(T o1, T o2);
```

```
Collections.sort(list, (x,y) -> y - 2 * x);
```

✓ `java.util.function.*` - набор функциональных интерфейсов общего назначения для разных случаев

- ☑ `Supplier<R> { R get() }`
- ☑ `Consumer<T> { void accept(T t) }`
- ☑ `Predicate<T> { boolean test(T t) }`
- ☑ `Function<T,R> { R apply(T t) }`
 - `UnaryOperator<T> { T apply(T t) }`
- ☑ `BiFunction<T,U,R> { R apply(T t, U u) }`
 - `BinaryOperator<T> { T apply(T t1, T t2) }`

```
Supplier<R> {  
    R get()  
}
```

- `IntSupplier { int getAsInt() }`
- `LongSupplier { long getAsLong() }`
- `DoubleSupplier { double getAsDouble() }`
- `BooleanSupplier { boolean getAsBoolean() }`

```
Consumer<T> {  
    void accept(T t)  
    default Consumer andThen(Consumer after)  
}
```

- `IntConsumer { void accept(int v) }`
- `LongConsumer { void accept(long v) }`
- `DoubleConsumer { void accept(double v) }`

```
Predicate<T> {  
    boolean test(T t)  
    default Predicate and(Predicate other)  
    default Predicate or(Predicate other)  
    default Predicate negate()  
}
```

- `IntPredicate { boolean test(int v) }`
- `LongPredicate { boolean test(long v) }`
- `DoublePredicate { boolean test(double v) }`

```
Function<T,R> {  
    R apply(T t)  
    default Function andThen(Function after)  
    default Function compose(Function before)  
    default Function identity()  
}
```

- `IntFunction<R> { R apply(int v) }`
- `doubleFunction<R> { R apply(double v) }`
- `ToLongFunction<T> { long applyAsLong(T t) }`
- `IntToLongFunction { long applyAsLong(int v) }`
- ...

```
UnaryOperator<T> {  
    T apply(T t)  
    default UnaryOperator andThen(UnaryOperator after)  
    default UnaryOperator compose(UnaryOperator before)  
    default UnaryOperator identity()  
}
```

- `IntUnaryOperator { int apply(int v) }`
- `LongUnaryOperator { long apply(long v) }`
- `DoubleUnaryOperator { double apply(double v) }`

BiConsumer

```
BiConsumer<T, U> {  
    void accept(T t, U u)  
    default BiConsumer andThen(BiConsumer after)  
}
```

```
ObjIntConsumer<T> { void accept(T t, int v) }  
ObjLongConsumer<T> { void accept(T t, long v) }  
ObjDoubleConsumer<T> { void accept(T t, double v) }
```

```
BiPredicate<T, U> {  
    boolean test(T t, U u)  
    default BiPredicate and(BiPredicate other)  
    default BiPredicate or(BiPredicate other)  
    default BiPredicate negate()  
}
```

BiFunction

```
BiFunction<T,U,R> {  
    R apply(T t, U u)  
    default BiFunction andThen(BiFunction after)  
}
```

```
ToIntBiFunction<T,U> { int applyAsInt(T t, U u) }  
ToLongBiFunction<T,U> { long applyAsLong(T t, U u) }  
ToDoubleBiFunction<T,U> { double applyAsDouble(T t, U u) }
```

BinaryOperator

```
BinaryOperator<T> {  
    T apply(T t, T t)  
    default BinaryOperator andThen(BinaryOperator after)  
    static BinaryOperator maxBy(Comparator comp)  
    static BinaryOperator minBy(Comparator comp)  
}
```

```
IntBinaryOperator { int applyAsInt(int v1, int v2) }  
LongBinaryOperator { long applyAsLong(long v1, long v2) }  
DoubleBinaryOperator { double applyAsDouble(double v1, double v2) }
```



УНИВЕРСИТЕТ ИТМО

Программирование. 2 семестр

Stream API

ITMO More than a
UNIVERSITY

- ☑ Конвейерная обработка данных
- ☑ Поток — последовательность элементов
- ☑ Поток может быть последовательным или параллельным
- ☑ Конвейер — последовательность операций

☑ Отличия конвейера от коллекции

- Элементы не хранятся
- Неявная итерация
- Функциональный стиль — операции не меняют источник
- Большинство операций работают с λ -выражениями
- Ленивое выполнение
- Возможность неограниченного числа элементов

☑ Конвейер =

- Источник
- Промежуточные операции (0 или больше)
- Завершающая операция (одна)

```
List<String> words
```

```
long count = 0;
for (String s : words) {
    if (s.length() > 5) {
        count++;
    }
}
```

```
long count =
    words.stream()
        .filter(s -> s.length() > 5)
        .count();
```

- ✓ `Collection.stream()`
- ✓ `Arrays.stream(Object[])`
- ✓ `Stream.of(Object[])`
- ✓ `IntStream.range(int, int)`
- ✓ `Files.lines(Path)`, `BufferedReader.lines()`
- ✓ `Random.ints()`
- ✓ `Stream.empty()`
- ✓ `Stream.generate(Supplier<T> s)`
- ✓ `Stream.iterate(T seed, UnaryOperator<T> f)`

- ✓ Возвращают поток
- ✓ Выполняются "лениво"
 - выполнение операции происходит, когда вызывается завершающая операция
- ✓ Делятся на:
 - Не хранящие состояние (stateless)
 - ◆ выполняются вне зависимости от других элементов
 - Хранящие состояние (stateful)
 - ◆ выполнение зависит от других элементов (сортировка)

- ✓ Возвращают результат
- ✓ Либо имеют побочное действие
- ✓ Поток прекращает существование.

☑ интерфейс BaseStream

- void close()
- S parallel()
- S sequential()
- S unordered()
- Iterator iterator()
- Spliterator spliterator()

☑ Интерфейс Stream<T>

☑ Интерфейсы IntStream, LongStream, DoubleStream

☑ Параллельный Iterator

- Spliterator trySplit() — возвращает часть элементов как отдельный сплитератор
- boolean tryAdvance(Consumer action) — выполнить операцию для очередного элемента
- void forEachRemaining(Consumer action) — выполнить операцию для всех оставшихся элементов

- ✓ Методы для промежуточных операций (stateless)
- ✓ **Stream<T> filter (Predicate<T> p)**
 - возвращает поток из элементов, соответствующих условию
- ✓ **Stream<R> map(Function<T,R> mapper)**
 - преобразует поток элементов T в поток элементов R
- ✓ **Stream<R> flatMap(Function <T, Stream<R>> mapper)**
 - преобразует каждый элемент потока T в поток элементов R
- ✓ **Stream<T> peek(Consumer<T> action)**
 - выполняет действие для каждого элемента потока T

- ✓ Методы для промежуточных операций (stateful)
- ✓ `Stream<T> distinct()`
 - возвращает поток неповторяющихся элементов
- ✓ `Stream<T> sorted(Comparator<T> comp)`
 - возвращает отсортированный поток
- ✓ `Stream<T> limit(long size)`
 - возвращает усеченный поток из size элементов
- ✓ `Stream<T> skip(long n)`
 - возвращает поток, пропустив n элементов

- ✓ `void forEach(Consumer<T> action)`
- ✓ `void forEachOrdered(Consumer<T> action)`
 - выполняет действие для каждого элемента потока
 - второй вариант гарантирует сохранение порядка элементов
- ✓ `Optional<T> min(), Optional<T> max()`
 - возвращают минимальный и максимальный элементы,
- ✓ `long count(), int (long, double) sum()`
 - возвращают количество и сумму элементов
- ✓ `OptionalInt, OptionalLong, OptionalDouble`
 - `int getAsInt(), long getAsLong(), double getAsDouble()`

- ✓ Оболочка: содержит или не содержит значение
- ✓ `boolean isPresent()` - true, если значение есть
- ✓ `T get()` - возвращает значение
- ✓ `Optional<T> of(T value)` — возвращает оболочку со значением
- ✓ `T orElse(T other)` — возвращает значение или other

- ☑ `reduce(BinaryOperator accumulator)`
 - `.stream`
 - `.reduce((a, b) -> a * b) // подсчет произведения`
- ☑ `collect(Collector collector)`
 - `.stream`
 - `.collect(Collectors.toList());`

- ☑ класс `Collectors`
 - `toCollection(Supplier factory), toList(), toSet()`
 - `toMap(Function k, Function v, BinaryOperator merge, Supplier factory)`
 - `joining(String delimiter, String prefix, String suffix)`
 - `mapping(Function<T,U> mapper, Collector<U> s)`
 - `minBy(Comparator), maxBy(Comparator)`
 - `counting(), summingDouble(), averagingDouble()`
 - `reducing(identity, Function<T,U> mapper, BinaryOperator op)`
 - `groupingBy(Function<T,K> classifier)`
 - `partitioningBy(Predicate<T> predicate)`

☑ boolean anyMatch(Predicate<T> p)

- истина, если условие выполняется хотя бы для одного элемента
- При нахождении первого совпадения прекращает проверку

☑ boolean allMatch(Predicate<T> p)

- истина, если условие выполняется для всех элементов.
- При нахождении первого несовпадения прекращает проверку

☑ boolean noneMatch(Predicate<T> p)

- истина, если условие не выполняется ни для одного элемента.
- При нахождении первого совпадения прекращает проверку

```
IntStream stream = IntStream
    .range(1, 5)
    .peek(System.out::println)
    .filter(i -> i % 2 == 0)
    .peek(i ->
        System.out.println(">> " + i));
System.out.println("Ready");
int c = stream.count();
System.out.println("Count: " + c);
```

```
IntStream stream = IntStream
    .range(1, 5)
    .peek(System.out::println)
    .filter(i -> i % 2 == 0)
    .peek(i ->
        System.out.println(">> " + i));
System.out.println("Ready");
int c = stream.count();
System.out.println("Count: " + c);
```

```
Ready
1
2
>> 2
3
4
>> 4
Count: 2
```

```
public static void main(String[] args) {  
    Arrays.asList(args).stream()  
        .filter(s -> s.length() < 5)  
        .map(String::toUpperCase)  
        .sorted()  
        .forEachOrdered(System.out::println);  
}
```