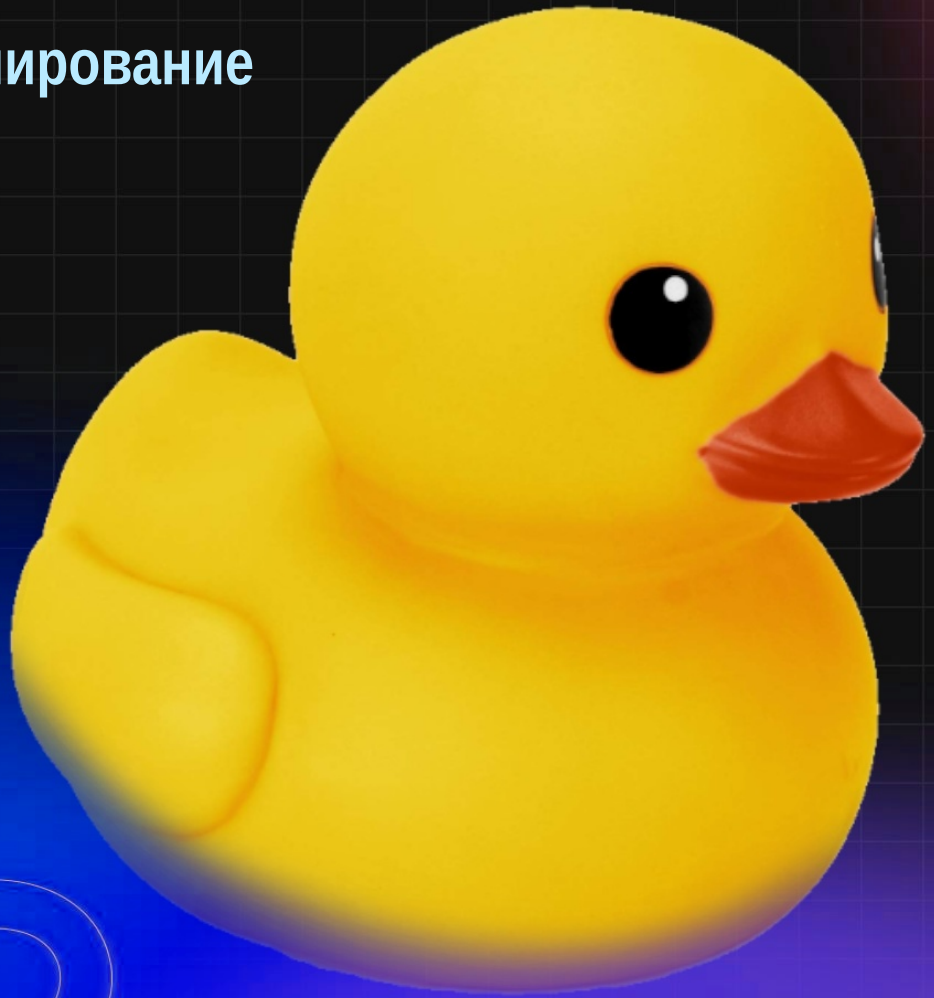
Программирование
2 семестр
2025

ИТМО



Ввод-вывод (NIO)

IO / NIO

ИТМО

- Преимущества NIO
  - поддержка буферов вне кучи (direct buffer)
  - поддержка неблокируемых каналов
  - поддержка мультиплексируемых каналов (selectors)
  - поддержка асинхронных каналов

Java

Блокируемый / неблокируемый I/O

ИТМО

- Блокируемый
  - Операция ввода-вывода блокирует выполнение
  - Нельзя продолжить, пока она не закончится
- Неблокируемый
  - Операция ввода-вывода возвращает -1, если нет данных
  - Можно сделать что-то еще, и попробовать еще раз

Java

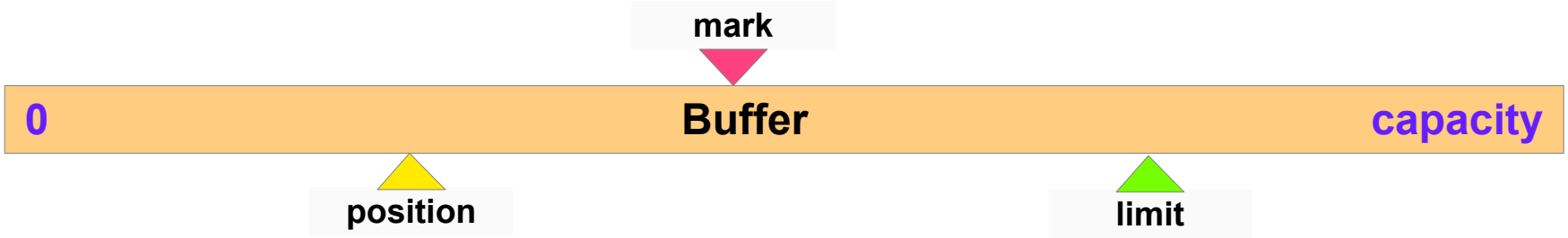
Синхронный / асинхронный I/O

ИТМО

- Синхронный
  - Операция ввода-вывода вернет данные в том же потоке
  - Последовательное выполнение: запрос - ответ - обработка
- Асинхронный
  - Операция ввода-вывода:
    - возвращает контейнер для будущего результата (Future)
    - получает объект-обработчик результата (CompletionHandler)
  - Не нужно ждать для отправки нового запроса

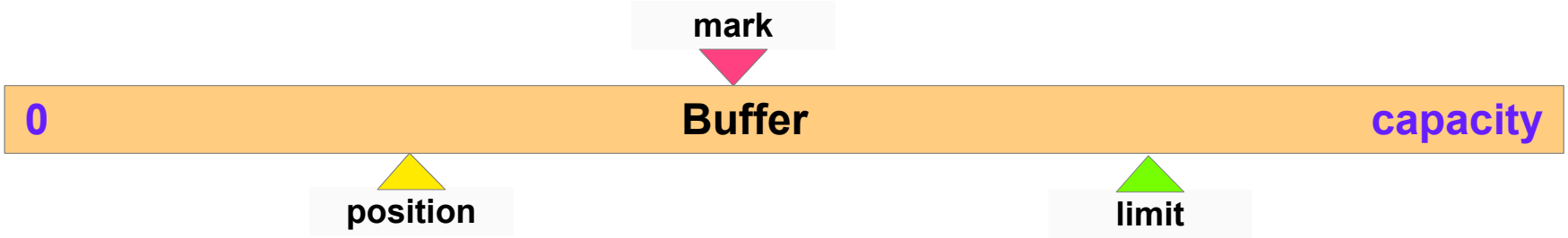
Java

java.nio - Buffer



- java.nio.Buffer — контейнер для хранения данных
  - capacity - емкость (максимальный размер)
  - limit - сколько можно записать (не больше емкости)
    или прочитать (не больше, чем записано)
  - position - текущая позиция
  - mark - метка

java.nio - Buffer



- 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

Запись и чтение данных

ИТМО

- 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

ИТМО

- **запись**

  clear();

  while () { put(byte); }

- чтение

  flip();

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

Java

Запись и чтение данных

ИТМО

- **запись**

  clear();

  while () { put(byte); }

- чтение

  flip();

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

Java

Запись и чтение данных

ИТМО

- **запись**

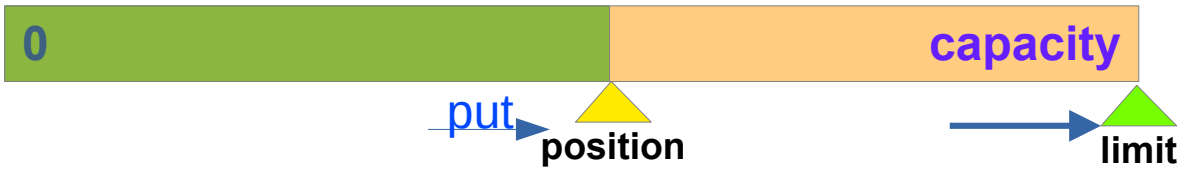
  clear();

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

- чтение

  flip();

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



Java

Запись и чтение данных

ИТМО

- **запись**

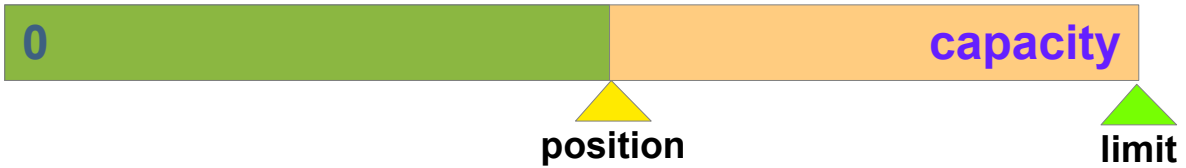
  clear();

  while () { put(byte); }

- чтение

  flip();

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



Java

Запись и чтение данных

ИТМО

- **запись**

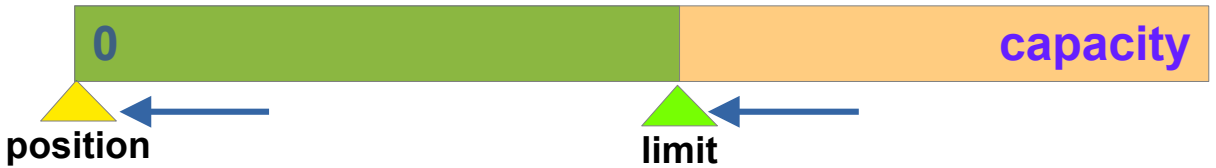
  clear();

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

- чтение

  flip();

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



Java

Запись и чтение данных

- запись

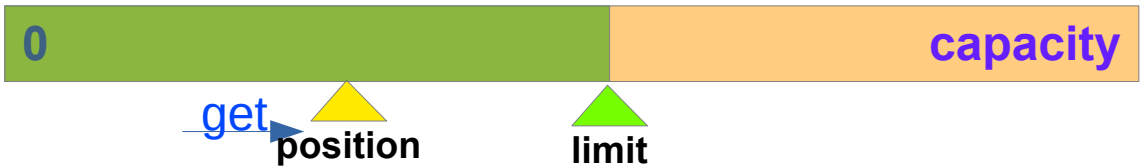
  clear();

  while () { put(byte); }

- **чтение**

  flip();

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



Классы-потомки Buffer

ИТМО

- ByteBuffer
- CharBuffer
- IntBuffer, ShortBuffer, LongBuffer, FloatBuffer, DoubleBuffer
  - getInt(), putInt(), getShort(), putShort() ...
- java.nio.ByteOrder
  - ByteOrder.BIG_ENDIAN
  - ByteOrder.LITTLE_ENDIAN
  - nativeOrder()
  - order()

Java

java.nio.charset

- Класс Charset - кодировка символов
  - методы
  - CharBuffer decode(ByteBuffer b)
  - ByteBuffer encode(CharBuffer c)

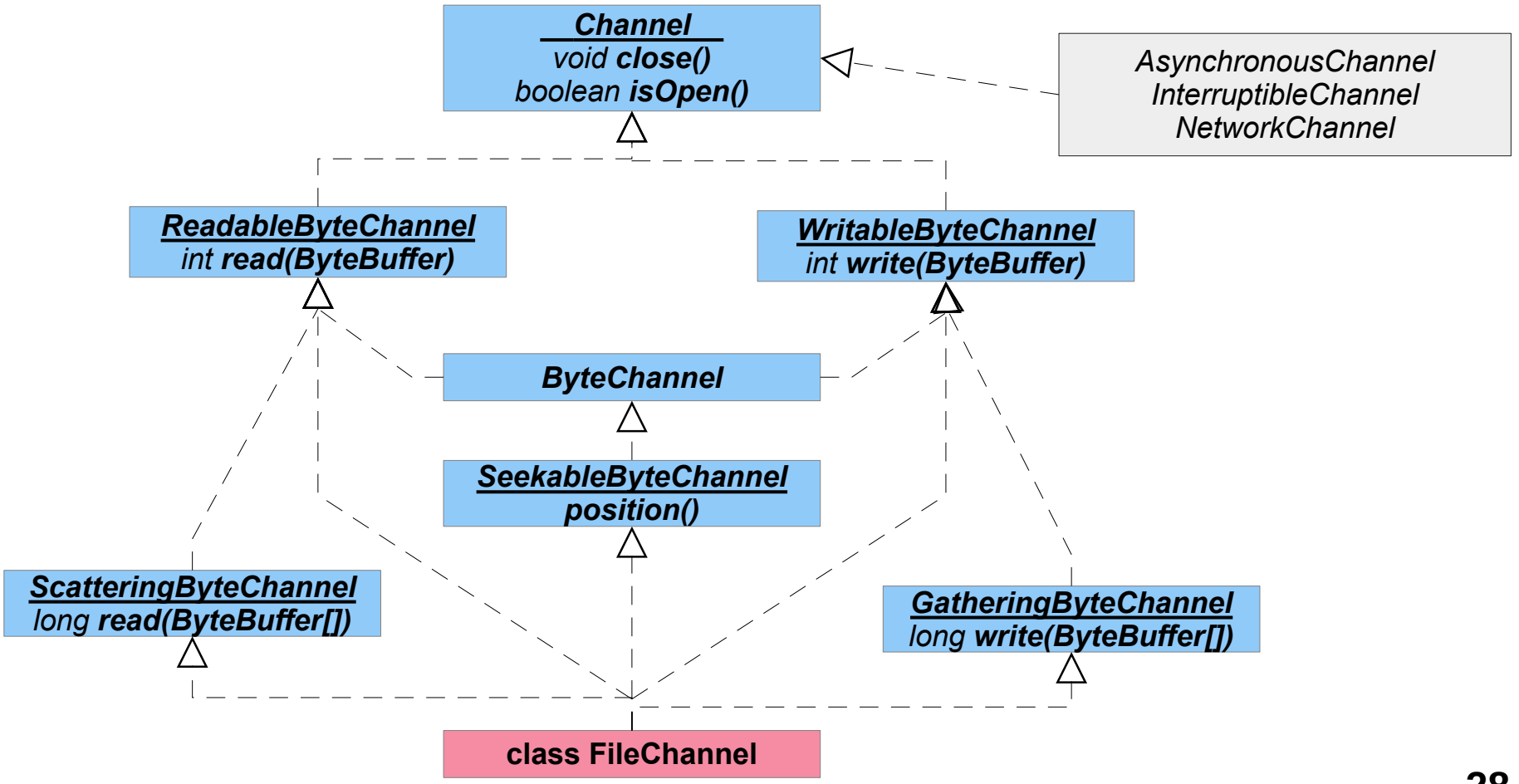
Каналы

ИТМО

- 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

Java

Каналы - интерфейсы и FileChannel



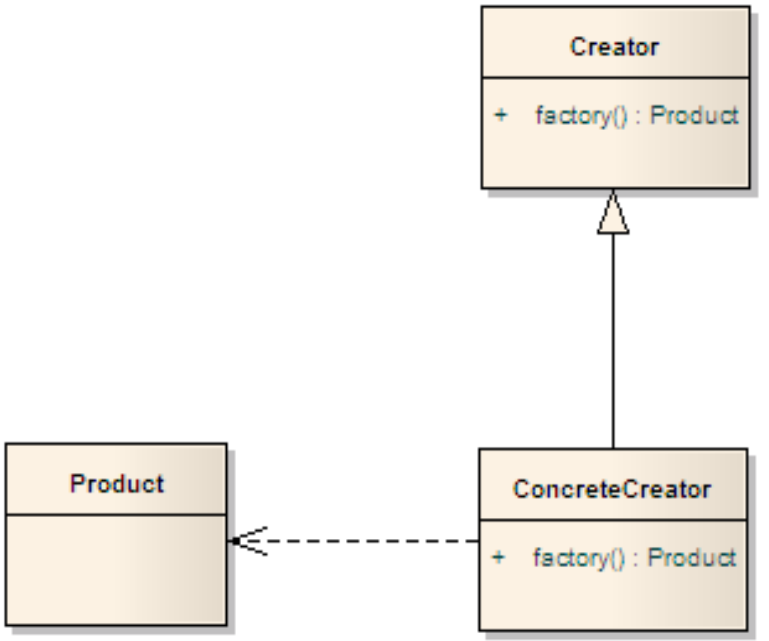
Шаблон Фабрика

- Фабрика и Продукт
- Создаем экземпляры Продукта с помощью Фабрики вместо вызова конструктора

Simple Factory

- Простая фабрика (не совсем паттерн)
  - Объект, создающий другие объекты

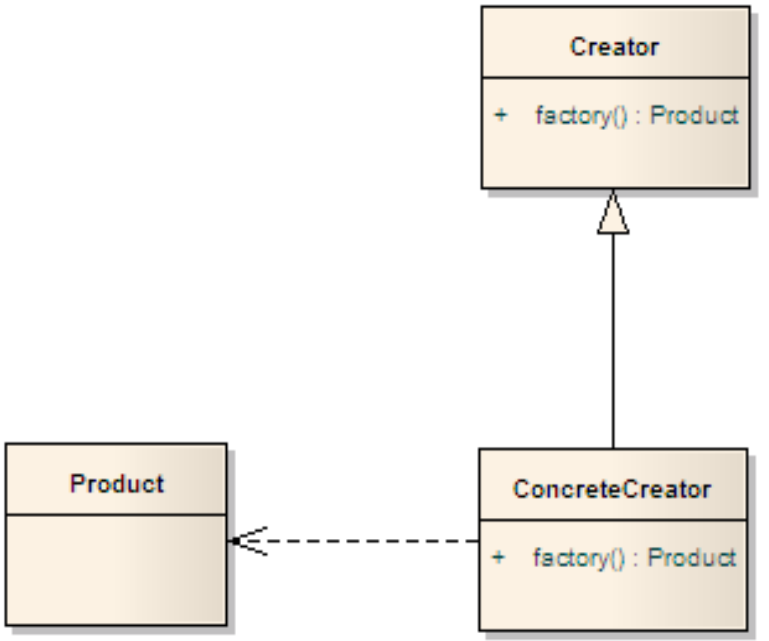
```
Animal[] animals = new Animal[2];
animals[0] = new Dog();
animals[1] = new Cat();
for (Animal a : animals) {
  a.sleep();
  a.makeSound();
}
```



Simple Factory

- Простая фабрика (не совсем паттерн)
  - Объект, создающий другие объекты

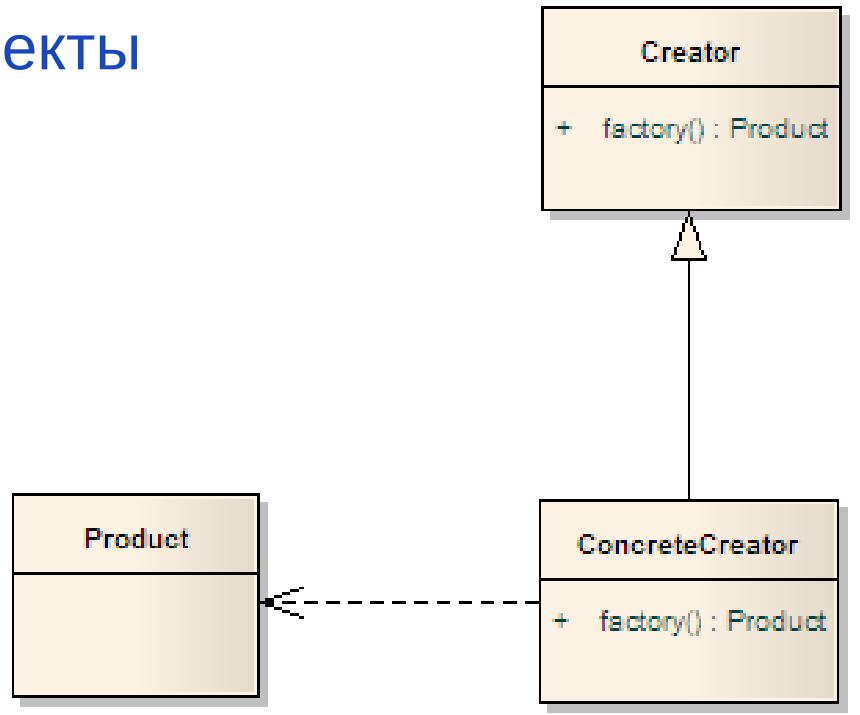
```
class Creator {
  public static Product get(int type) {
    return switch(type) {
      case 1 -> product1;
      case 2 -> product2;
  }
}
```



Simple Factory

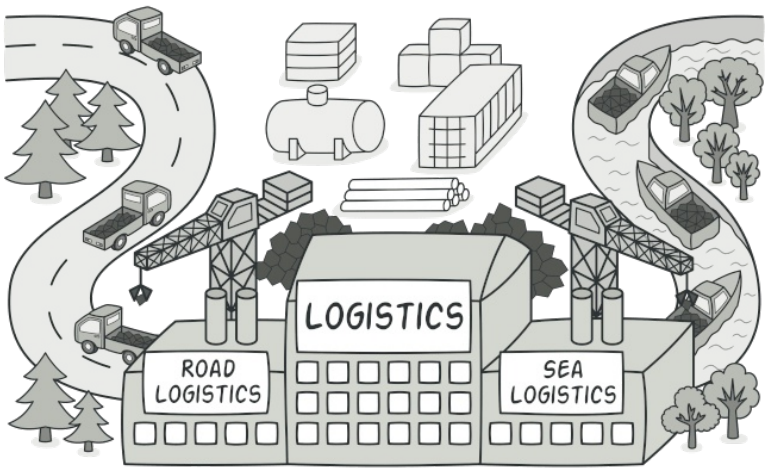
- Простая фабрика (не совсем паттерн)
  - Объект, создающий другие объекты

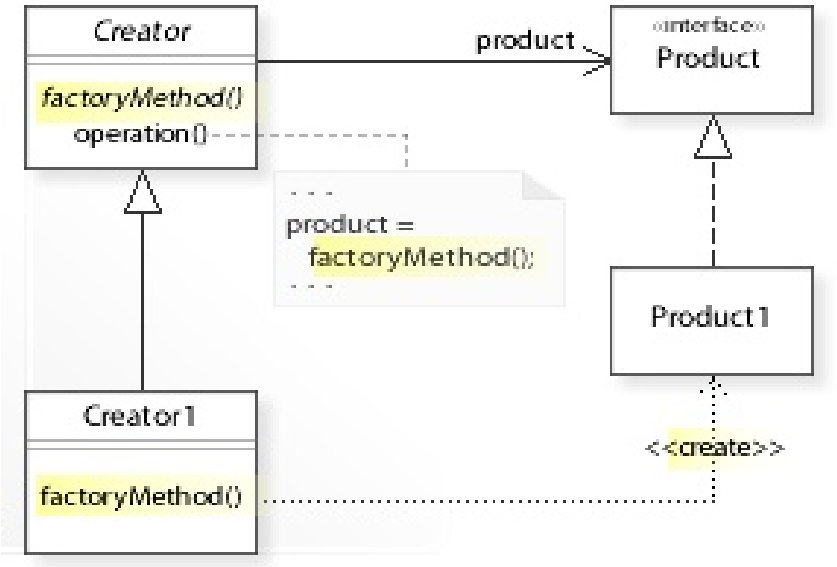
```
Animal[] animals = new Animal[2];
animals[0] = animalFactory.get(1);
animals[1] = animalFactory.get(2);
for (Animal a : animals) {
  a.sleep();
  a.makeSound();
}
```



Factory Method

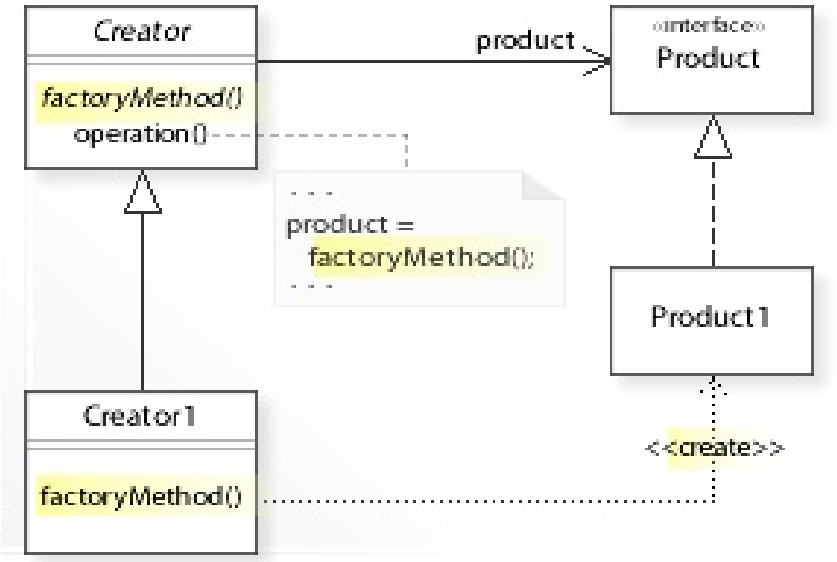
- Фабричный метод
  - Создание объектов отдаем подклассам





Factory Method

- Фабричный метод
  - Создание объектов отдаем подклассам
- Product - общий интерфейс
- Product1 - конкретный продукт
- Creator - интерфейс фабрики
  - Product factoryMethod()
  - другие методы...
- Creator1 - создает Product1



Factory Method

- Фабричный метод
  - Statement stat = connection.createStatement()

```
class Creator {
  protected abstract Product getProduct();
}

class Creator1 {
  public Product getProduct() {
    return new Product1();
  }
}
```

Factory Method

- Убирает зависимость от конкретных продуктов
- Упрощает добавление новых продуктов
- На каждый продукт нужен свой создатель

☑ Канал FileChannel

ИТМО

- фабричные методы:
  - FileChannel.open()
  - FileInputStream.getChannel()
- write(ByteBuffer b)
  - запись в канал из буфера
- read(ByteBuffer b)
  - чтение из канала в буфер

Java

Чтение и запись

```
ByteBuffer buffer =
ByteBuffer.allocate(1000);
```

- Чтение из файла (файлового канала)

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

- Запись в файл (файловый канал)

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

Продвинутые каналы

ИТМО

- 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);
```

Java

Отображение файла в память

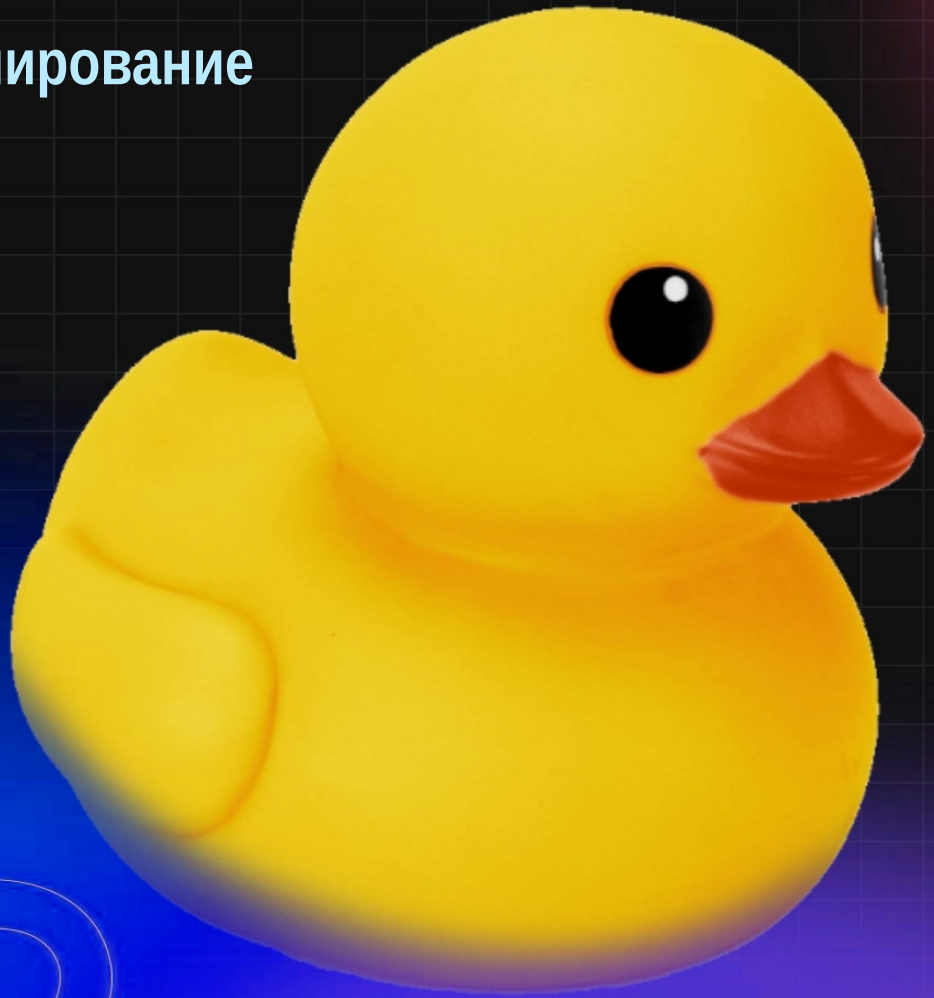
ИТМО

- FileChannel
  - map() - получение MappedByteBuffer
  - отображение файла в память
- MappedByteBuffer
  - boolean isLoaded()
  - load()
  - force()

```
var mode = FileChannel.MapMode.READ_ONLY;
var buffer =
    channel.map(mode, 0, fileChannel.size());
```

Java

Программирование
2 семестр
2025

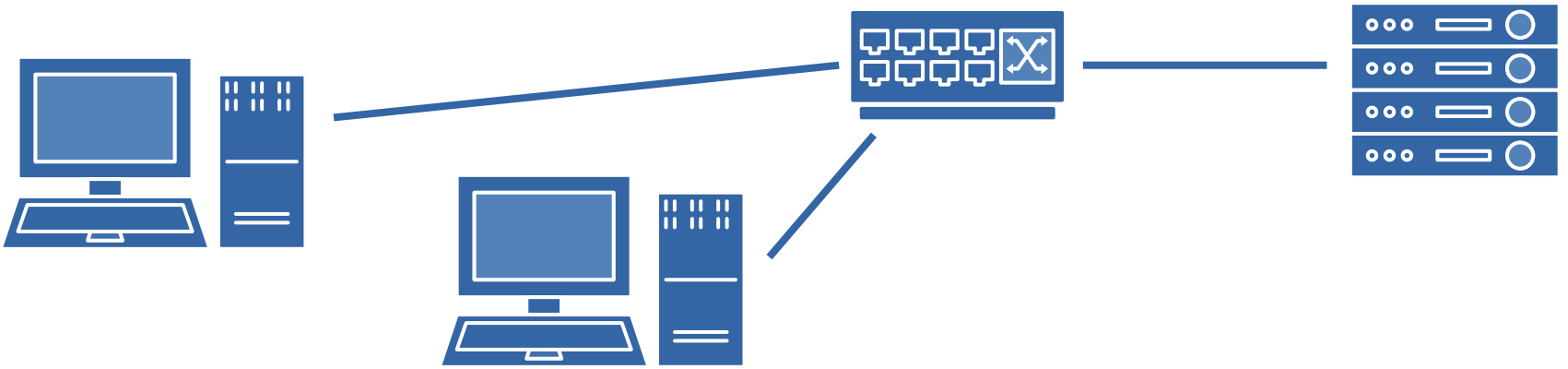


ИТМО

Сетевое взаимодействие

Терминология

- *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.



Сетевая архитектура

ИТМО

- *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 (pears) равноправны, могут быть клиентом и сервером
  - Нет критического узла

Модель передачи данных

ИТМО

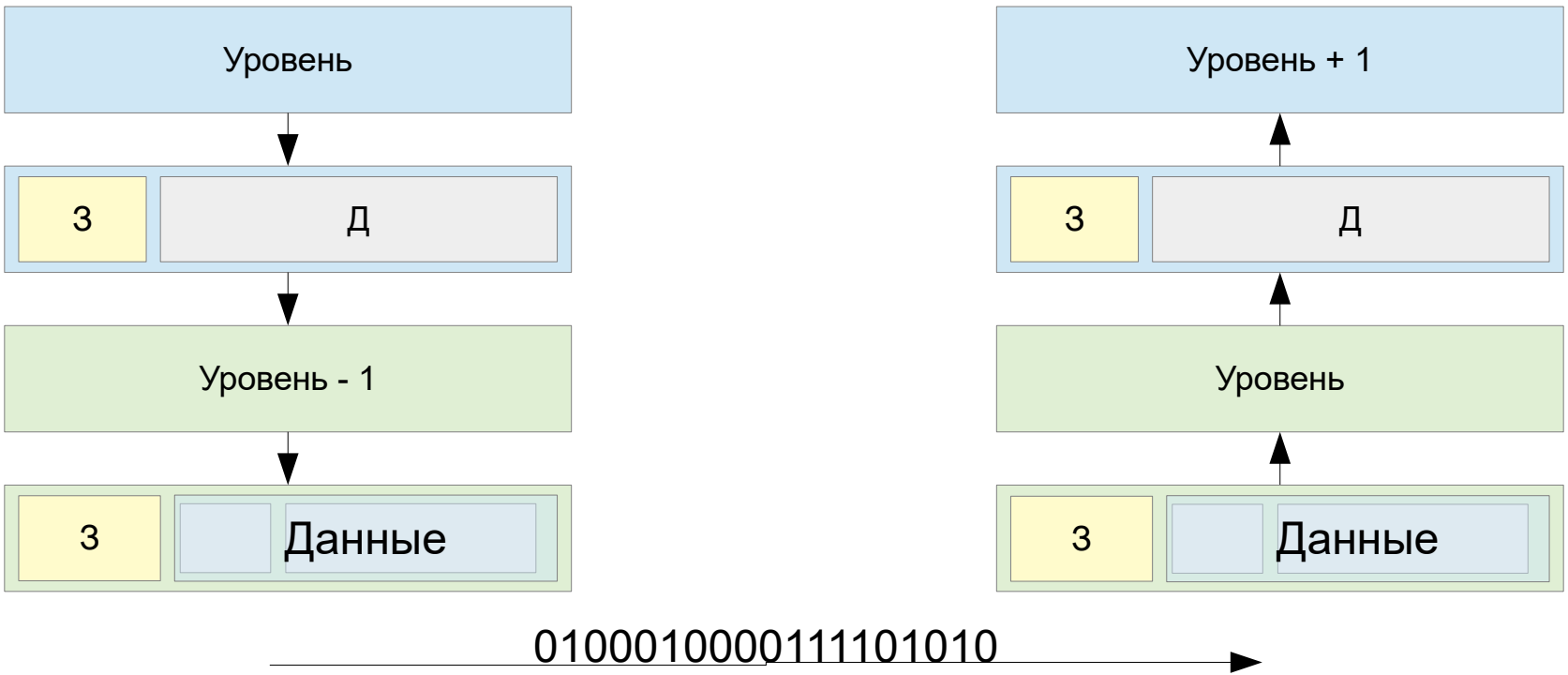
- модель ISO/OSI
- стек TCP/IP

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

Java

Пакеты

- Пакет = заголовок + данные



Сетевое взаимодействие в Java

ИТМО

- Протокол прикладного уровня - HTTP

- Протоколы транспортного уровня
- TCP
  - устанавливается соединение
  - подтверждение доставки
  - надежность передачи данных
- UDP
  - без установление соединения
  - без подтверждения доставки
  - скорость передачи данных

Java

IP-адрес

- Идентифицирует связь между роутером и хостом
- ID сети (префикс) + ID хоста (суффикс)
- IPv4 — 32 бита (194.85.160.55)
  - Класс A: префикс 8 бит (0...) + суффикс 24 бита
  - Класс B: префикс 16 бит (10…) + суффикс 16 бит
  - Класс C: 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

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

Порт

ИТМО

- 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

Java

InetAddress

- Класс 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()

InetSocketAddress

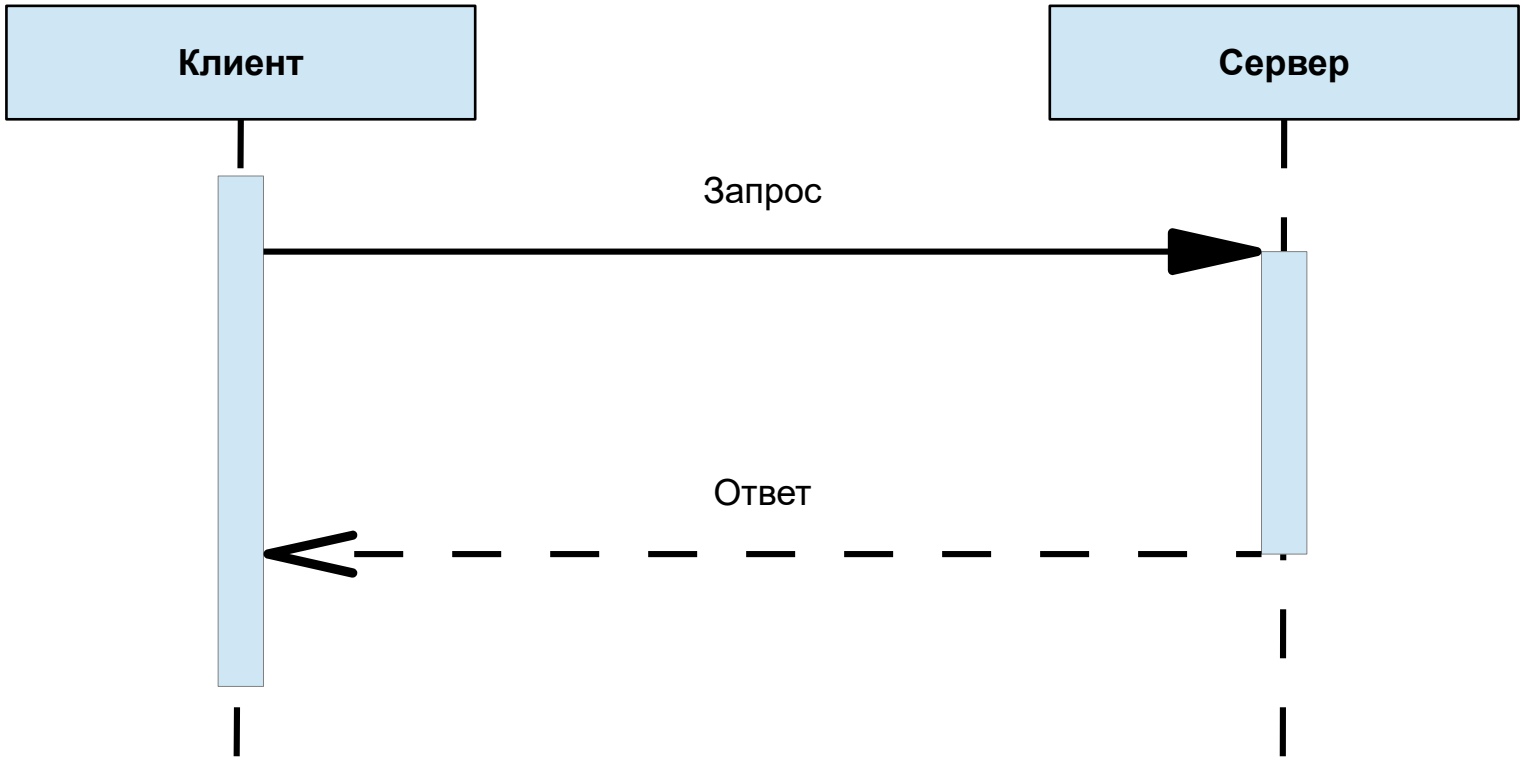
- InetSocketAddress(InetAddress addr, int port)
- InetSocketAddress(int port)
- InetSocketAddress(String hostname, int port)

Сетевой обмен

ИТМО

- 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



Обмен по протоколу UDP

ИТМО

- java.net.DatagramPacket — датаграмма (передаваемые данные + служебная информация)
  - Адрес буфера
  - Длина буфера (не более 64 Кбайт)
  - Адрес получателя (при отправлении)
- java.net.DatagramSocket — сокет для обмена
  - Порт для прослушивания (для получения)
  - Адрес и порт (для отправления)

Java

Пример обмена по протоколу UDP

ИТМО

```
// 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

Обмен по протоколу TCP

ИТМО

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

Java

Пример обмена по протоколу TCP

ИТМО

```
// 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);
```

Java

Использование каналов и java.nio

ИТМО

- 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)

ИТМО

```
// 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);
```

Java

Использование каналов и java.nio

ИТМО

- 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)

ИТМО

```
// 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);
```

Java

Режимы обмена

- Блокирующий режим
  - Можно ли выполнить операцию = попытаться выполнить
    - setSoTimeout(long milliseconds)
    - Socket / ServerSocket / DatagramSocket
- Неблокирующий режим
  - Проверка возможности - отдельно от самой операции

Неблокирующий ввод-вывод

ИТМО

- ServerSocketChannel.configureBlocking(false)
  - Метод accept() возвращает SocketChannel или null, если соединение не установлено
- SocketChannel.configureBlocking(false)
  - Методы read и write возвращают int — количество прочитанных байт, или -1, если данных больше нет.

Java

Селекторы

ИТМО

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

Java

Селекторы - ключи

ИТМО

- abstract class SelectionKey
  - interestOps(), readyOps()
  - OP_CONNECT, OP_ACCEPT, OP_READ, OP_WRITE
  - isConnectable(), isAcceptable(), isReadable(), isWriteable()
  - channel(), selector()
  - attach(Object), Object attachment()
  - cancel()

Java

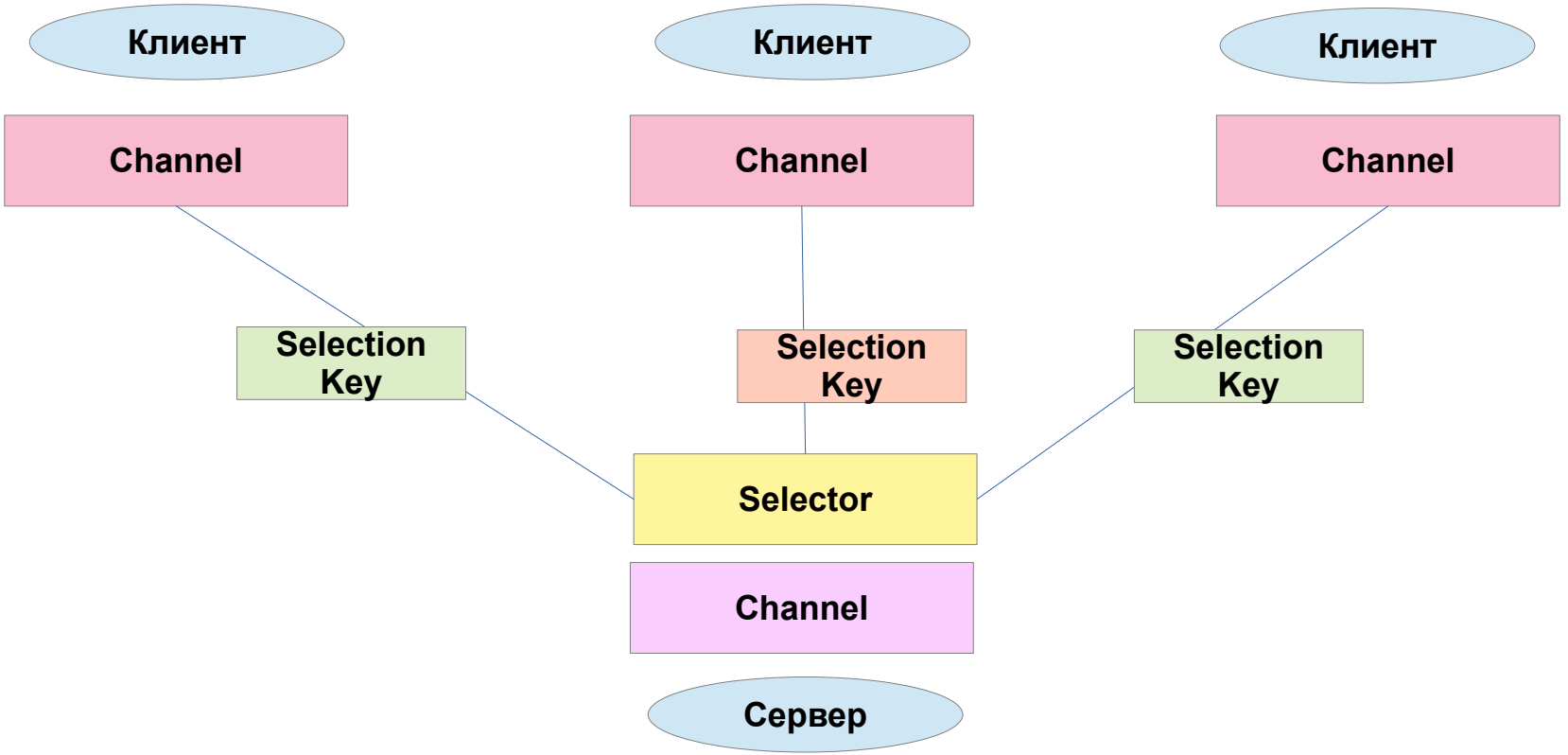
Селекторы - каналы

ИТМО

- abstract class SelectableChannel
  - SelectonKey register(Selector s, int Ops, Object attachment)
    - SocketChannel
    - ServerSocketChannel
    - DatagramChannel

Java

Селекторы и каналы



Пример

```
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

ИТМО

```
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);
}
```

Протокол HTTP

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

Класс URL

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

InputStream is = url.openStream();
is.read();
is.close();

Object o = url.getContent();
```

Класс URLConnection

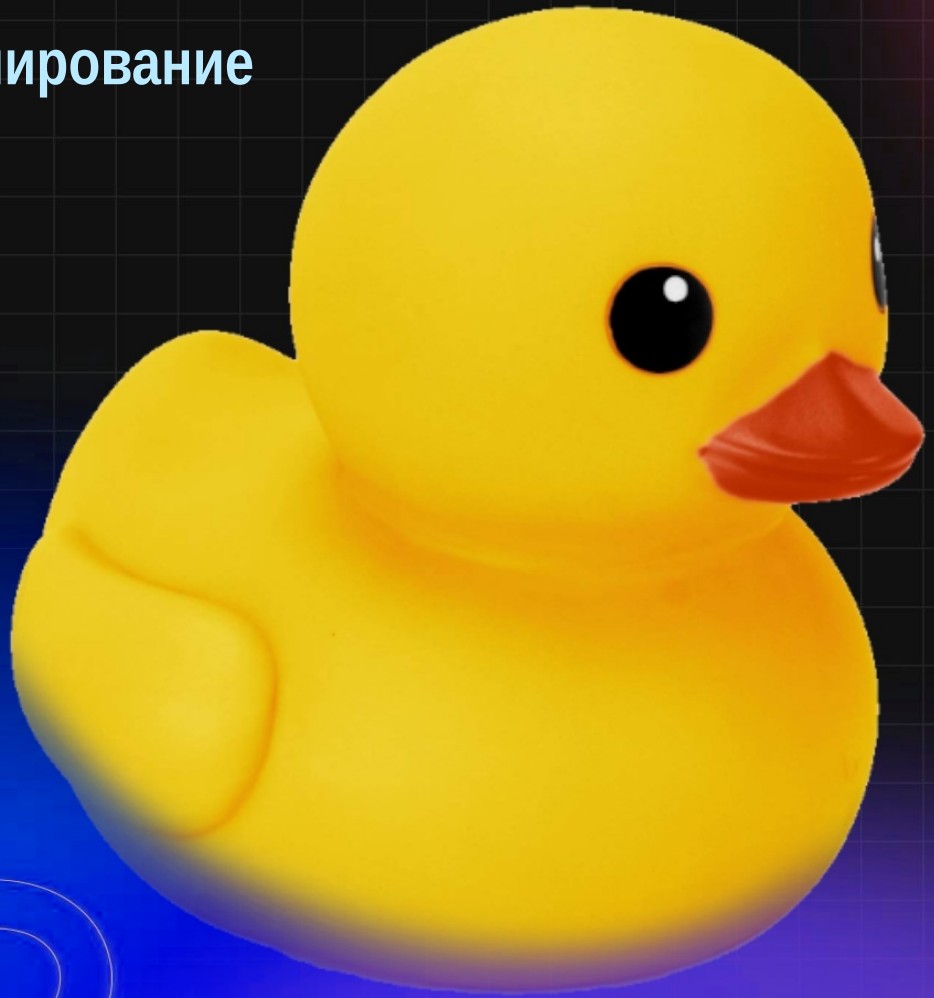
ИТМО

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

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

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

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

Java

Программирование
2 семестр
2025

ИТМО



Функциональное программирование

Функциональное программирование

ИТМО

- Функции высшего порядка
  - могут быть аргументами и возвращаемыми значениями
- Ленивые вычисления
- Нет побочных эффектов
- Нет состояния

- Достоинства:
  - Проще тестирование
  - Проще распараллеливание
  - Оптимизация кода

Java

Итерация и рекурсия

ИТМО

- Итерация → Рекурсия
- Проблема рекурсии — 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

Java

Итерация и рекурсия

ИТМО

- Итерация → Рекурсия
- Проблема рекурсии — 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)

Java

Факториал (итерация)

ИТМО

- Итерация

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

Java

Факториал (рекурсия)

ИТМО

- Итерация

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

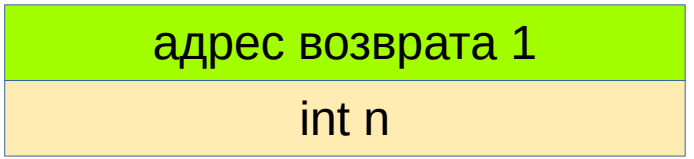
- Рекурсия
  - проще код
  - проблема стека вызовов

```
public int factor(int n) {
  return n <= 1 ?
         1 :
         factor(n – 1) * n;
}
```

Java

Стек вызовов при рекурсии

- 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

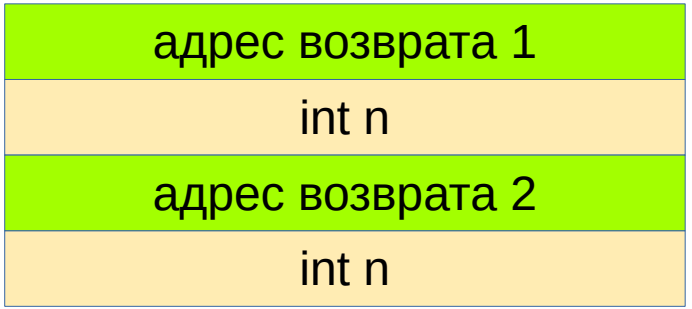



```
public int factor(int n) {
  return n <= 1 ?
         1 :
         factor(n – 1) * n;
}
```

Стек вызовов при рекурсии

ИТМО

- 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



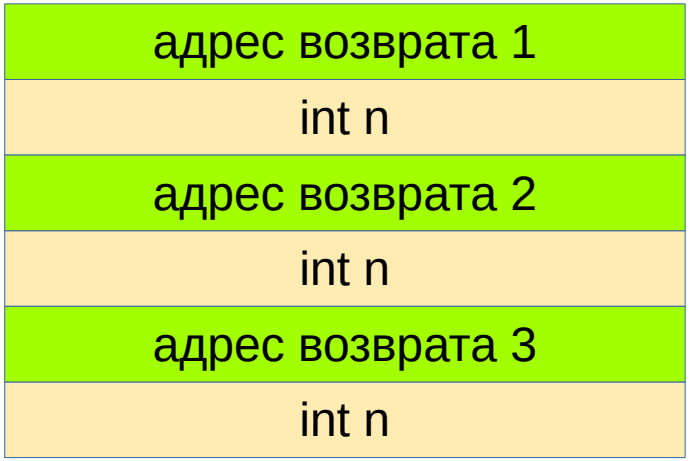

```
public int factor(int n) {
  return n <= 1 ?
         1 :
         factor(n – 1) * n;
}
```

Java

Стек вызовов при рекурсии

ИТМО

- 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



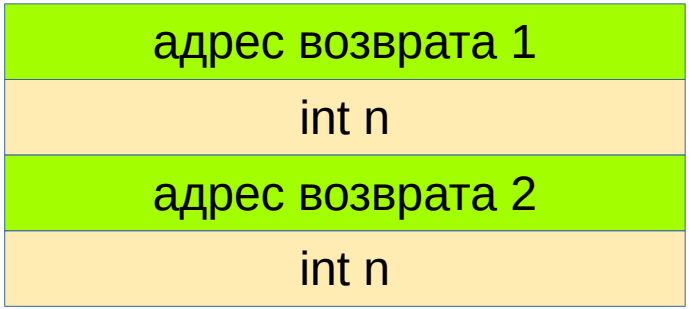

```
public int factor(int n) {
  return n <= 1 ?
         1 :
         factor(n – 1) * n;
}
```

Java

Стек вызовов при рекурсии

ИТМО

- 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) {
  return n <= 1 ?
         1 :
         factor(n – 1) * n;
}
```

Java

Факториал (хвостовая рекурсия)

ИТМО

- Хвостовая рекурсия

```
public int factor(int n, int p) {
  return n <= 1 ?
         p :
         factor(n – 1, p * n);
  }
}

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

- Обычная рекурсия

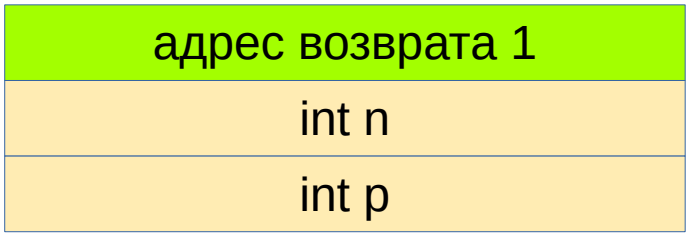
```
public int factor(int n) {
  return n <= 1 ?
         1 :
         factor(n – 1) * n;
}
```

Java

Факториал (хвостовая рекурсия)

- Хвостовая рекурсия

```
public int factor(int n, int p) {
  return n <= 1 ?
         p :
         factor(n – 1, p * n);
  }
}

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



Факториал (хвостовая рекурсия)

ИТМО

- Хвостовая рекурсия

```
public int factor(int n, int p) {
  return n <= 1 ?
         p :
         factor(n – 1, p * n);
  }
}

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

адрес возврата 1
адрес возврата 2
int n
int p

Java

Стек при хвостовой рекурсии

ИТМО

- Хвостовая рекурсия

```
public int factor(int n, int p) {
  return n <= 1 ?
         p :
         factor(n – 1, p * n);
  }
}

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

адрес возврата 1
int n
int p

Java

Элементы λ-исчисления

ИТМО

- Алонзо Чёрч (Alonzo Church)
- â.a+1 → λa.a+1 → Λa.a+1 → λa.a+1
  - â — аргумент выражения a+1
- Переменная: x
- Операции:
  - Абстракция: λx.f (связывание x с функцией f)
  - Аппликация (применение): f g (применение f к аргументу g)

Java

Пример

- inc(x) = x + 1
- f(x) = x + 1
- (x) → x + 1
- λx.x+1

inc(3)

f(3)

((x) → x + 1) (3)

- Свободные и связанные переменные

λ x . x + y

связанная

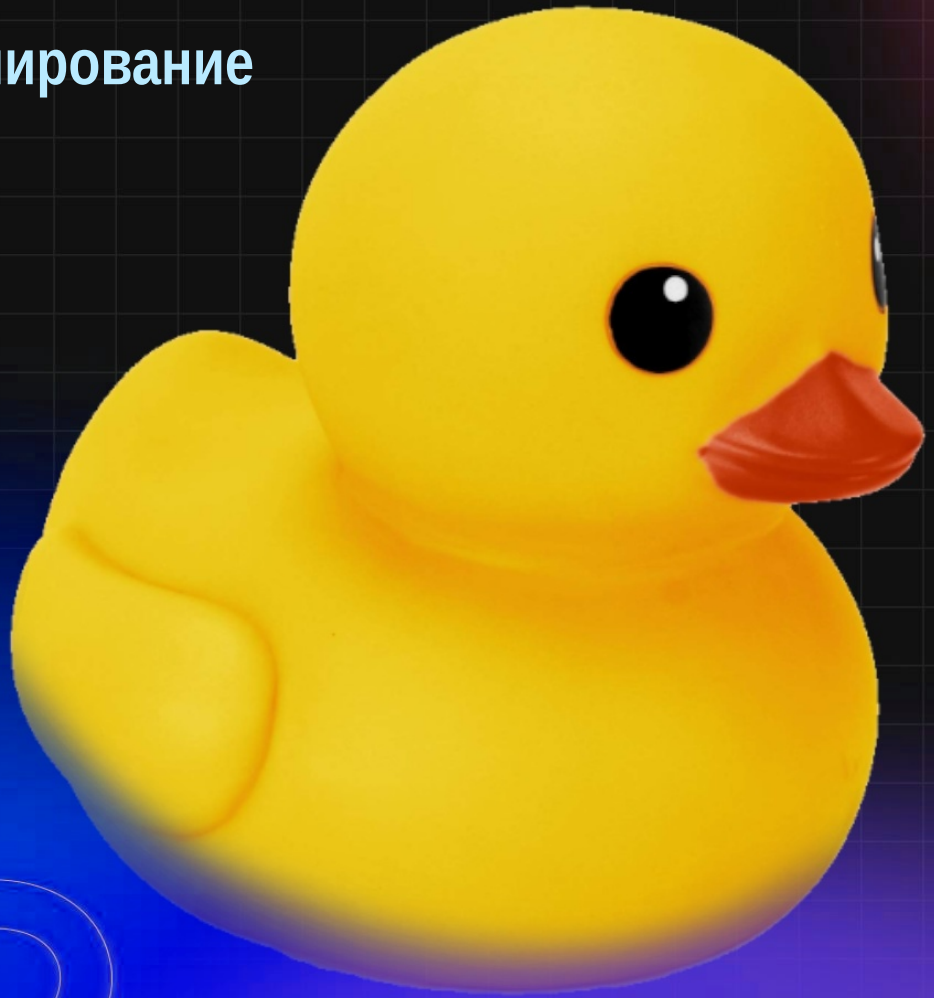
свободная

Функции обратного вызова (callback)

ИТМО

- 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+)

Программирование
2 семестр
2025



Практические примеры

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());
    }
  }
}
```

Список отличников группы

ИТМО

```
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());
      }
    }
  }
}
```

Java

Список молодых отличников

ИТМО

```
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());
      }
    }
  }
}
```

Java

Список избранных :)

ИТМО

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

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

Java

Интерфейс и класс для отбора

ИТМО

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

Java

Анонимный класс

ИТМО

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

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

Java

Функциональный интерфейс

ИТМО

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

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

Java

Лямбда-выражение

ИТМО

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

Java

Лямбда-выражение

ИТМО

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

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

Java

Лямбда-выражение

ИТМО

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

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

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

Java

java.util.function.*

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

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

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

Predicate - условие

ИТМО

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

class Student {
  ...
  public static 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"));
```

Java

+ Consumer - принять результат

ИТМО

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

class Student {
  ...
  public static 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));
```

Java

+ Iterable - универсальный обработчик

ИТМО

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

class Student {
  ...
  public static <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));
```

Java

+ Iterable - универсальный обработчик

ИТМО

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

class Student {
  ...
  public static <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));
```

Java

Ссылка на метод (Method Reference)

ИТМО

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

class Student {
  ...
  public static <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));
```

Java

Ссылка на метод (Method Reference)

ИТМО

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

class Student {
  ...
  public static <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);
```

Java

Варианты ссылок на метод

ИТМО

- Аналог лямбда-выражения с вызовом метода
- Синтаксис: **`TypeName::methodName`**
  - Type::staticMethod
  - object::instanceMethod
  - Class::instanceMethod
  - Type.super::instanceMethod
  - Class::new
  - type[]::new
  - x -> Type.staticMethod(x)
  - x -> object.instanceMethod(x)
  - (Class x) -> x.instanceMethod()
  - x -> Type.super.instanceMerhod(x)
  - x -> new Class(x)
  - x -> new type[x]

@FunctionalInterface и λ

ИТМО

- 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

Java

@FunctionalInterface и λ

ИТМО

- λ-выражение можно присвоить переменной
  - Comparator<Integer> comp = (x, y) -> y - x;
- λ-выражение можно передать методу
  - **public static <T> void sort(T[] a, Comparator<? super T> c)**
  - Arrays.sort(array, (s1, s2) -> s1.length() - s2.length);
  - Arrays.sort(array2, comp);
- λ-выражение можно вернуть из метода
  - **public static <T> Comparator<T> reverseOrder()**
  - Arrays.sort(array3, Collections.reverseOrder());

Синтаксис λ-выражений

ИТМО

```
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; }
```

Java

Захват переменных

ИТМО

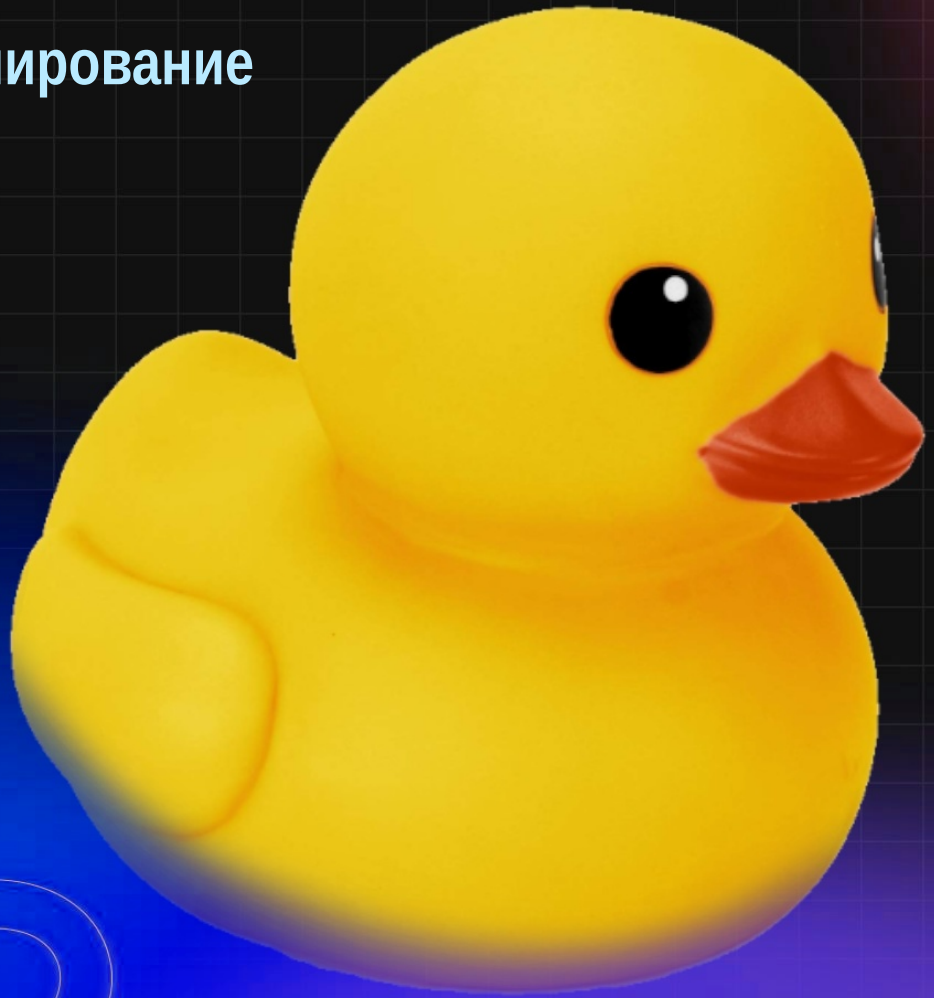
- Область видимости λ-выражения = область видимости окружающего блока
- В λ-выражении можно использовать только эффективно финальные переменные из окружающего его блока
- λ-выражение захватывает значения переменных из окружающего блока.
- λ-выражение + значения захваченных переменных = замыкание (closure)

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

Java

Программирование
2 семестр
2025

ИТМО



Функциональные интерфейсы в Java API

Функциональные интерфейсы

ИТМО

- 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

Пакет java.util.function

ИТМО

- `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) }`

Java

Supplier

ИТМО

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

```
void Logger.log(Level level, Supplier<String> msgSupplier)
```

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

Consumer

ИТМО

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

void ArrayList.forEach(Consumer<? super E> action)
```

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

Predicate

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

boolean ArrayList.removeIf(Predicate<? super E> filter)
```

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

Function

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

<R> R String.transform(Function<? super String,? extends R> f)
```

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

UnaryOperator

ИТМО

```
UnaryOperator<T> extends Function<T,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

ИТМО

```
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

```
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) }
```

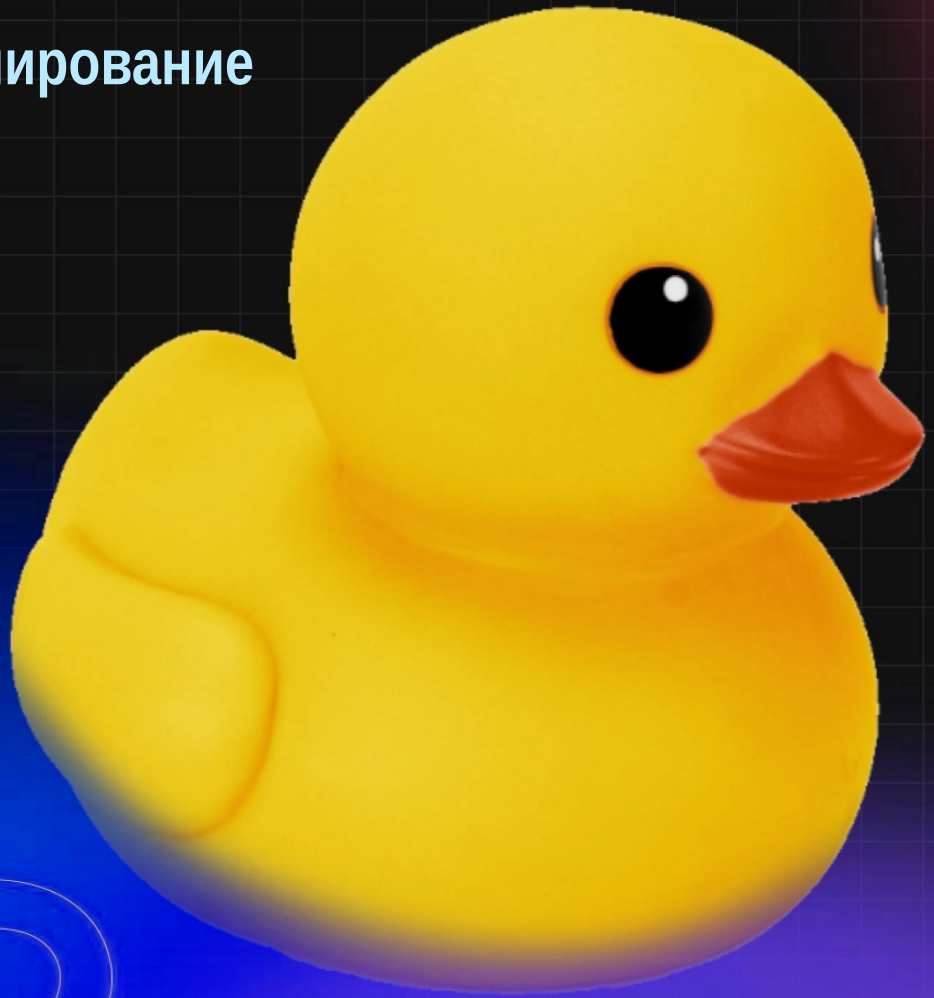
BinaryOperator

ИТМО

```
BinaryOperator<T> extends BiFunction<T,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) }
```

Программирование
2 семестр
2025

ИТМО



Stream API

Пакет java.util.stream

ИТМО

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

Java

Конвейеры и коллекции

ИТМО

- 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

Java

Состав конвейера

ИТМО

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

Java

Пример

```
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();
```

Источники конвейера

ИТМО

- `Collection.stream()`
- `Arrays.stream(T[] array), .stream(int[] array)`
- `Stream.of(T values)`
- `Stream.Builder.add(T element).build()`
- `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)`

Промежуточные операции

ИТМО

- Возвращают поток
- Выполняются "лениво"
  - Конвейер не запускается, пока нет завершающей операции
- Делятся на:
  - Не хранящие состояние (stateless)
    - выполняются вне зависимости от других элементов
  - Хранящие состояние (stateful)
    - выполнение зависит от других элементов (сортировка)

Завершающие операции

ИТМО

- Запускают конвейер
- Возвращают результат
- Либо имеют побочное действие
- Поток прекращает существование

Java

Классы и интерфейсы

ИТМО

- интерфейс BaseStream
  - void close()
  - S parallel()
  - S sequential()
  - S unordered()
  - Iterator iterator()
  - Spliterator spliterator()
- Интерфейс Stream<T>
- Интерфейсы IntStream, LongStream, DoubleStream

Spliterator

ИТМО

- 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

Промежуточные операции (stateless)

ИТМО

- **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

Java

Промежуточные операции (stateful)

ИТМО

- 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

Java

Завершающие операции

ИТМО

- 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 элементо
- OptionalInt, OptionalLong, OptionalDouble
  - int getAsInt(), long getAsLong(), double getAsDouble()

Класс java.util.Optional<T>

ИТМО

- 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
- void ifPresent(Consumer<T> action) - выполняет действие, если есть значение

Java

Завершающие операции

ИТМО

- `T reduce(T identity, BinaryOperator<T> accumulator)`

```
.stream
.reduce((a, b) -> a * b)  // подсчет произведения
```

- `R collect(supplier, accumulator, combiner)`
- `R collect(Collector<? super T,A,R> collector)`

```
.stream
.collect(Collectors.toList());
```

- `Object[] toArray()`
- `List toList()`

Java

Завершающие операции

ИТМО

- интерфейс Collector<T,A,R>
  - T - входные элементы, R - результат, A - аккумулятор
- 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)

Java

Проверки

ИТМО

- 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

Java

Ленивое выполнение

ИТМО

```
var list = new ArrayList<>(List.of(1,2,3));
var stream = list
    .stream()
    .peek(System.out::println)
    .filter(i -> i % 2 == 0)
    .peek(i -> System.out.println("> "+i));

System.out.println("Ready");
list.add(4);
long count = stream.count();
System.out.println("Count: " + count);
```

Ленивое выполнение

```
var list = new ArrayList<>(List.of(1,2,3));
var stream = list
    .stream()
    .peek(System.out::println)
    .filter(i -> i % 2 == 0)
    .peek(i -> System.out.println("> "+i));

System.out.println("Ready");
list.add(4);
long count = stream.count();
System.out.println("Count: " + count);
```

Ленивое выполнение

ИТМО

```
var list = new ArrayList<>(List.of(1,2,3));
var stream = list
    .stream()
    .peek(System.out::println)
    .filter(i -> i % 2 == 0)
    .peek(i -> System.out.println("> "+i));

System.out.println("Ready");
list.add(4);
long count = stream.count();
System.out.println("Count: " + count);
```

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

Java

Пример

ИТМО

```
public static void main(String[] args) {
  List sortedArgs =
    Arrays.stream(args)
          .filter(s -> s.length() < 5)
          .map(String::toUpperCase)
          .sorted()
          .collect(Collectors.toList());
}
```