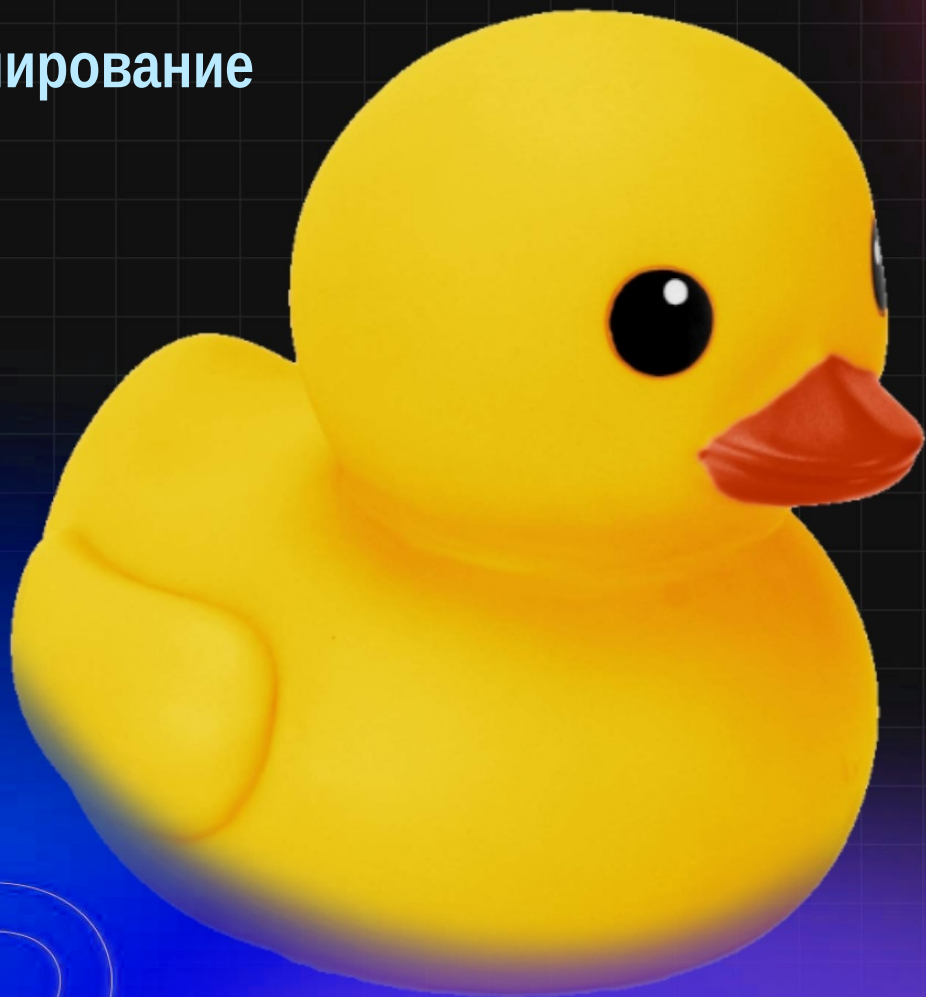


Программирование
2 семестр
2025



ІТМО

Сравнение
элементов

- Интерфейсы Comparable<T> и Comparator<T>
 - ❖ основной метод возвращает int (< 0, == 0, > 0)
- java.lang.Comparable<T> (x.compareTo(y))
 - ❖ естественный порядок сортировки
 - ❖ реализуется при создании класса
 - ❖ реализован в большинстве библиотечных классов
- java.util.Comparator<T> (c.compare(x, y))
 - ❖ любой необходимый порядок сортировки
 - ❖ объект создается по мере необходимости

- 1) Выбираем два элемента
 - 2) Как-то сравниваем - Comparable или Comparator
 - 3) При необходимости перемещаем элементы
- Готовые эффективные алгоритмы сортировки
 - ❖ List.sort() - stable, adaptive, iterative mergesort
 - Все, что нужно - предоставить метод сравнения

```
public class A {  
    List<String> list;  
    public void do() {  
        Collections.sort(list);  
        for (String s : list) { System.out.println(s); }  
    }  
}
```

class String implements Comparable

Сортировка (отдельный компаратор)

```
public class A {  
    List<String> list;  
    public void do() {  
        Collections.sort(list, new C());  
        for (String s : list) { System.out.println(s); }  
    }  
}  
  
class C implements Comparator<String> {  
    public int compare(String s1, String s2) {  
        return s1.length() - s2.length();  
    }  
}
```



Сортировка (вложенный static класс)

```
public class A {  
    List<String> list;  
    public void do() {  
        Collections.sort(list, new C());  
        for (String s : list) { System.out.println(s); }  
    }  
    static class C implements Comparator<String> {  
        public int compare(String s1, String s2) {  
            return s1.length() - s2.length();  
        }  
    }  
}
```



Сортировка (внутренний класс)

```
public class A {  
    List<String> list;  
    public void do() {  
        Collections.sort(list, new A().new C());  
        for (String s : list) { System.out.println(s); }  
    }  
    class C implements Comparator<String> {  
        public int compare(String s1, String s2) {  
            return s1.length() - s2.length();  
        }  
    }  
}
```

Сортировка (локальный класс)

```
public class A {  
    List<String> list;  
    public void do() {  
        class C implements Comparator<String> {  
            public int compare(String s1, String s2) {  
                return s1.length() - s2.length();  
            }  
        }  
        Collections.sort(list, new C());  
        for (String s : list) { System.out.println(s); }  
    }  
}
```

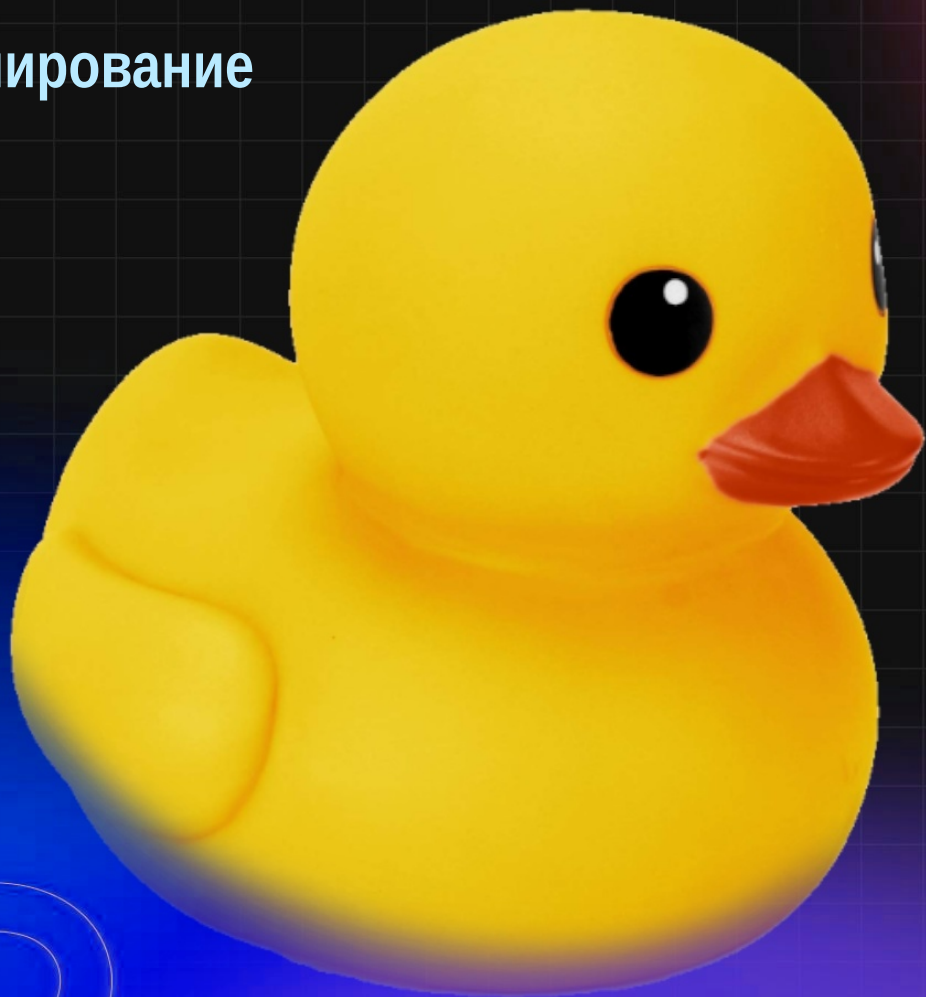


Сортировка (анонимный класс)

```
public class A {  
    List<String> list;  
    public void do() {  
        Collections.sort(list, new Comparator<String>() {  
            public int compare(String s1, String s2) {  
                return s1.length() - s2.length();  
            }  
        });  
        for (String s : list) { System.out.println(s); }  
    }  
}
```

```
public class A {  
    List<String> list;  
    public void do() {  
        Collections.sort(list,  
            (s1, s2) -> s1.length() - s2.length());  
        for (String s : list) { System.out.println(s); }  
    }  
}
```

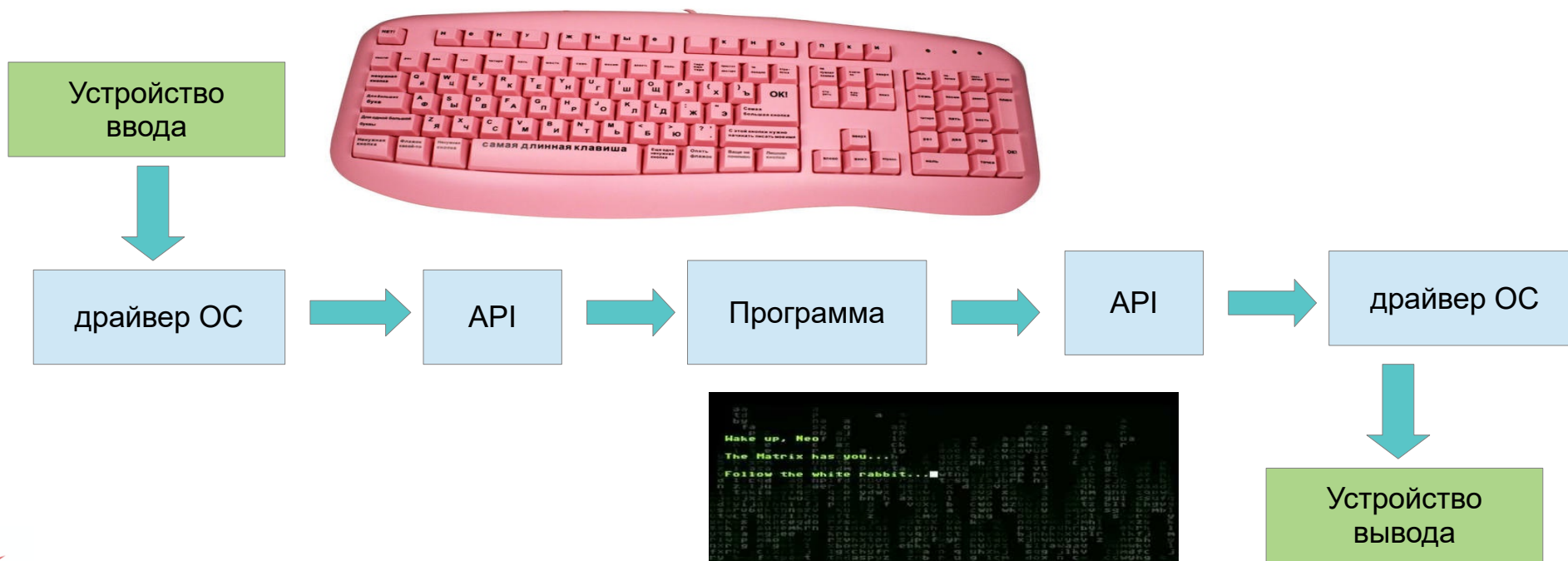
Программирование
2 семестр
2025



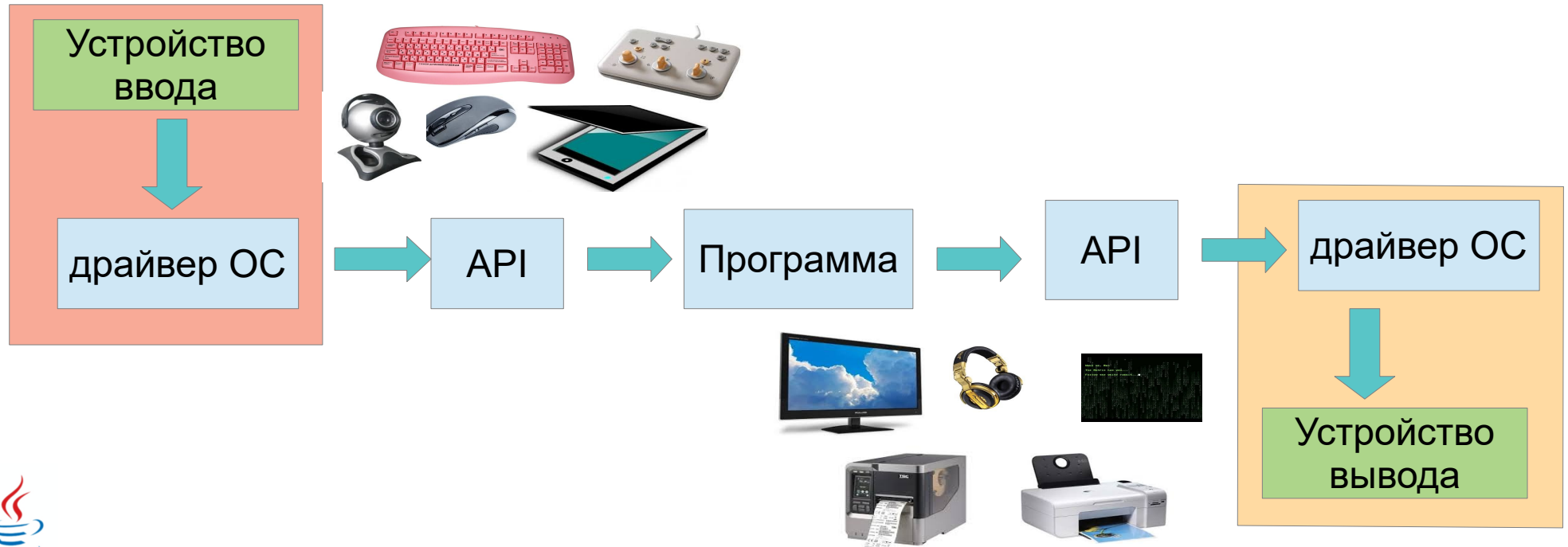
ІТМО

ВВОД-ВЫВОД

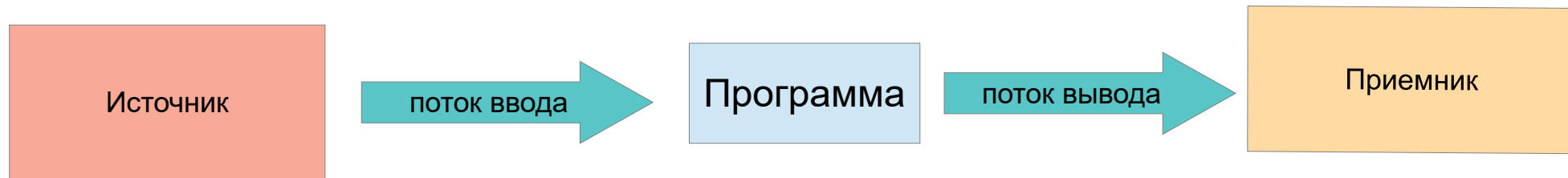
- java.io
 - ❖ Абстракция потока ввода-вывода
 - ❖ Данные - поток байтов/символов
- java.nio
 - ❖ Абстракция канала и буфера
 - ❖ Буфер - хранение данных
 - ❖ Канал - соединение для передачи данных



- Потоки ввода-вывода
- Источник данных и приемник данных



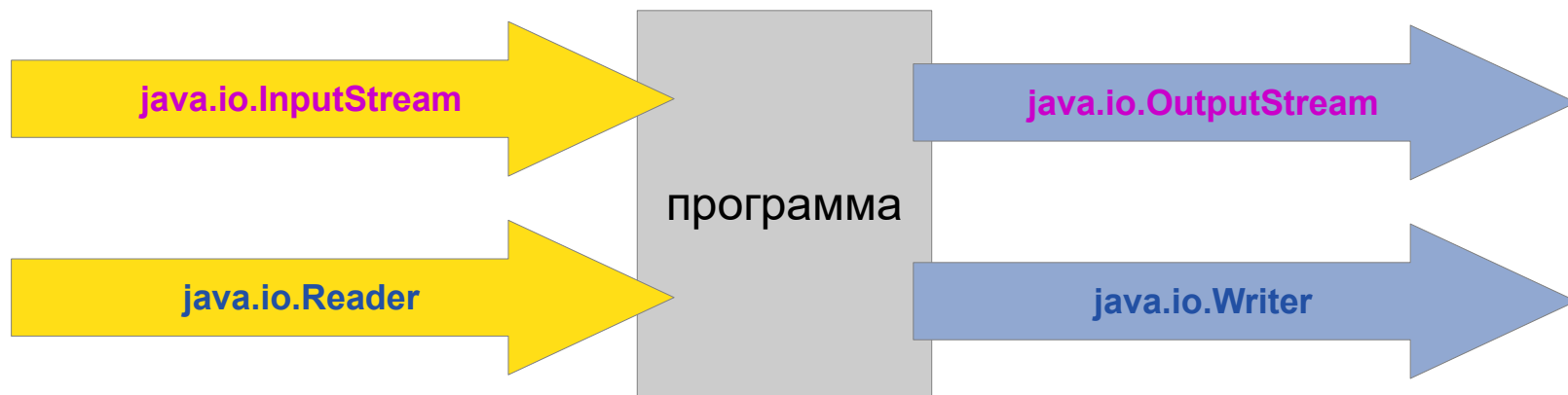
- Потоки ввода-вывода
- Источник данных и приемник данных



- Байтовые и символьные потоки данных (I/O streams)
 - ❖ Поток — последовательность данных (байтов, символов, примитивных типов, объектов)
 - ❖ Поток ввода — для чтения данных из источника
 - ❖ Поток вывода — для записи данных в приемник
- Старый интерфейс работы с файлами — класс File


```
Integer number = 51966;
00000000 00000000 11001010 11111110 // big-endian
11111110 11001010 00000000 00000000 // little-endian
number.toString() // "51966" =
'5' '1' '9' '6' '6'
00000000 00110101 00000000 00110001 00000000 00111001
00000000 00110110 00000000 00110110
number.toHexString() // "cafe"
'c' 'a' 'f' 'e'
00000000 01100011 00000000 01100001 00000000 01100110
00000000 01100101
UTF-8 — BOM (byte order mark) '\uFEFF'
```

- Базовые абстрактные классы для потоков
- Ввода и вывода
- Байтовые и символьные



- abstract **int** read()

- ❖ прочитанный байт

- ❖ -1 (конец потока)

00000000	00000000	00000000	10100111
11111111	11111111	11111111	11111111

- int read(byte[] buf, int off, int len)

- int available()

- long skip(long n)

- void **close()**

- boolean markSupported()

- void mark(int limit)

- void reset()

- abstract **int** read(char[] buf, int off, int len)

- ❖ количество байт
- ❖ -1 (конец потока)

- **int** read()

- ❖ read(buf, 0, 1)

- **int** available()

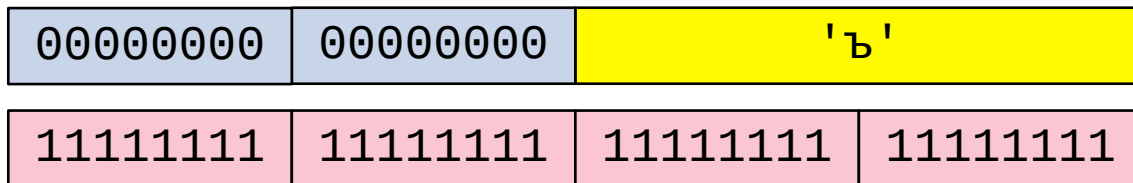
- **long** skip(long n)

- **void** close()

- **boolean** markSupported()

- **void** mark(int limit)

- **void** reset()



- abstract void write(int b)

❖ записываемый байт



- void write(byte[] buf, int off, int len)

- void flush()

- void close()

- abstract write(char[] buf, int off, int len)

- void write(int c)

 - ❖ write(buf, 0, 1)



- void flush()

- void close()

- Writer append(int c)

- Writer append(CharSequence cs)

- `flush()` - очистка внутреннего кэша или буфера
- данные сливаются в приемник
 - ❖ После выполнения метода внутренние буферы и кэши должны быть пустыми, данные переданы операционной системе для записи

- `close()` throws `IOException` - освобождение ресурса

```
try {  
    InputStream ins = new InputStream();  
    ins.read();  
} finally {  
    if (ins != null) ins.close();  
}
```

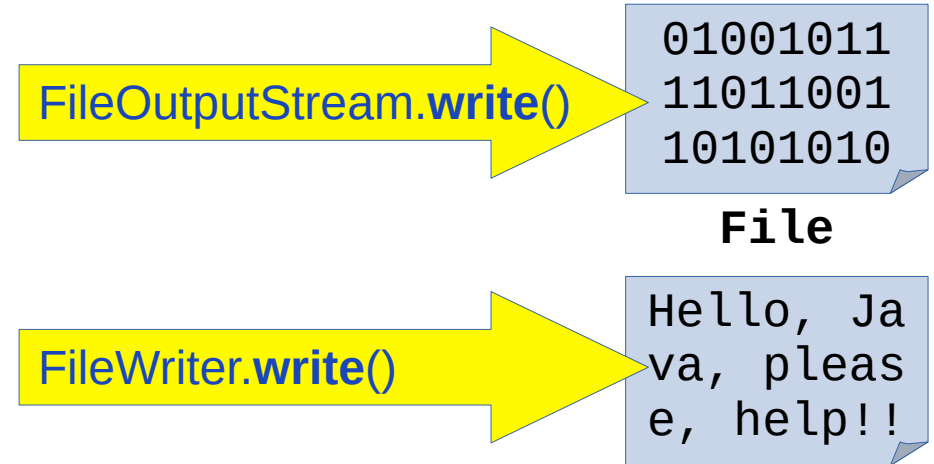
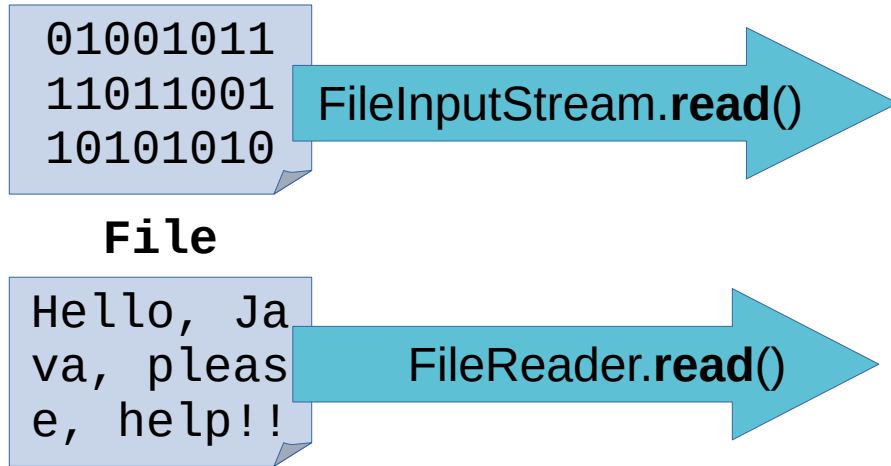

- Closable extends AutoCloseable
- close() вызывается автоматически
- блок try с ресурсом

```
try(InputStream ins = new InputStream()) {  
    ins.read();  
}
```

```
InputStream ins = new InputStream();  
try(ins) {  
    ins.read();  
}
```

Java 9

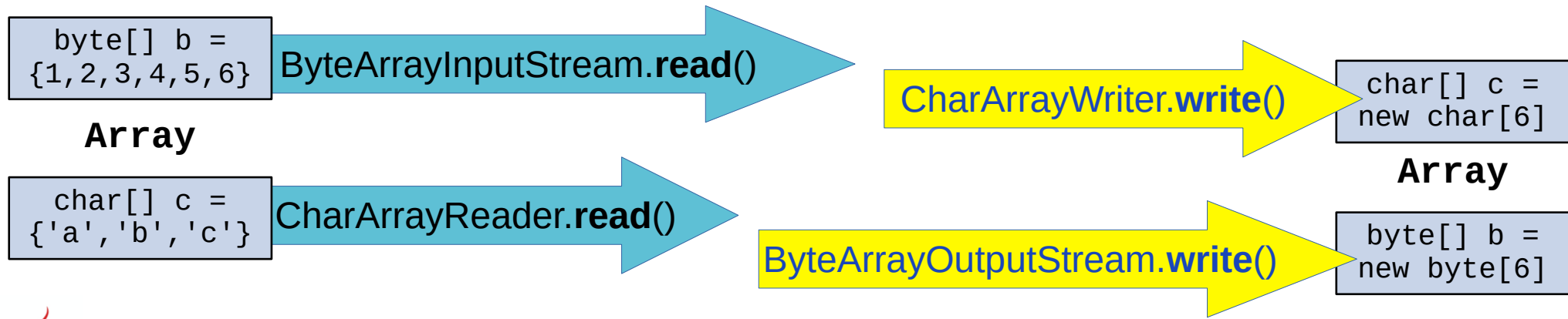
- `FileOutputStream`
- `FileWriter`
- `FileInputStream`
- `FileReader`



```
FileInputStream in = new FileInputStream("in.bin");
FileOutputStream out = new FileOutputStream("out.bin");
try (in, out) {
    int c;
    while ((c = in.read()) != -1) {
        out.write(c);
    }
    out.flush();
} catch (IOException e) {
    System.err.println(e);
}
```

```
try (FileReader in = new FileReader("in.txt");  
    FileWriter out = new FileWriter("out.txt")) {  
    int c;  
    while ((c = in.read()) != -1) {  
        out.write(c);  
    }  
    out.flush();  
} catch (IOException e) {  
    System.err.println(e);  
}
```

- `ByteArrayOutputStream`
 - ❖ `toByteArray()`
- `CharArrayWriter`
 - ❖ `toCharArray()`
- `ByteArrayInputStream`
- `CharArrayReader`



- **StringWriter**

- ❖ `StringBuffer getBuffer()`
- ❖ `String toString()`

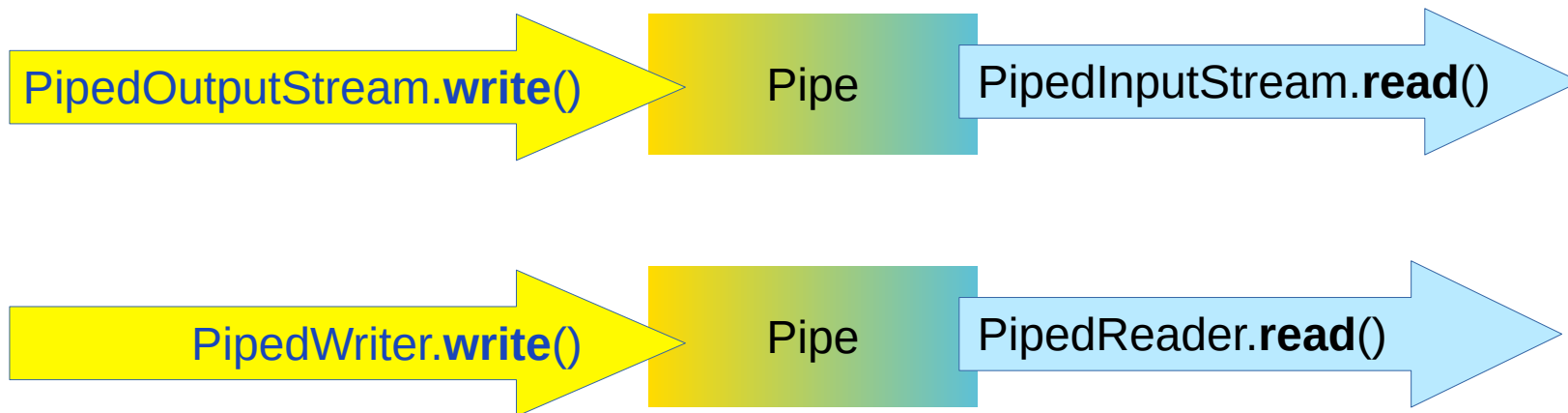
- **StringReader**



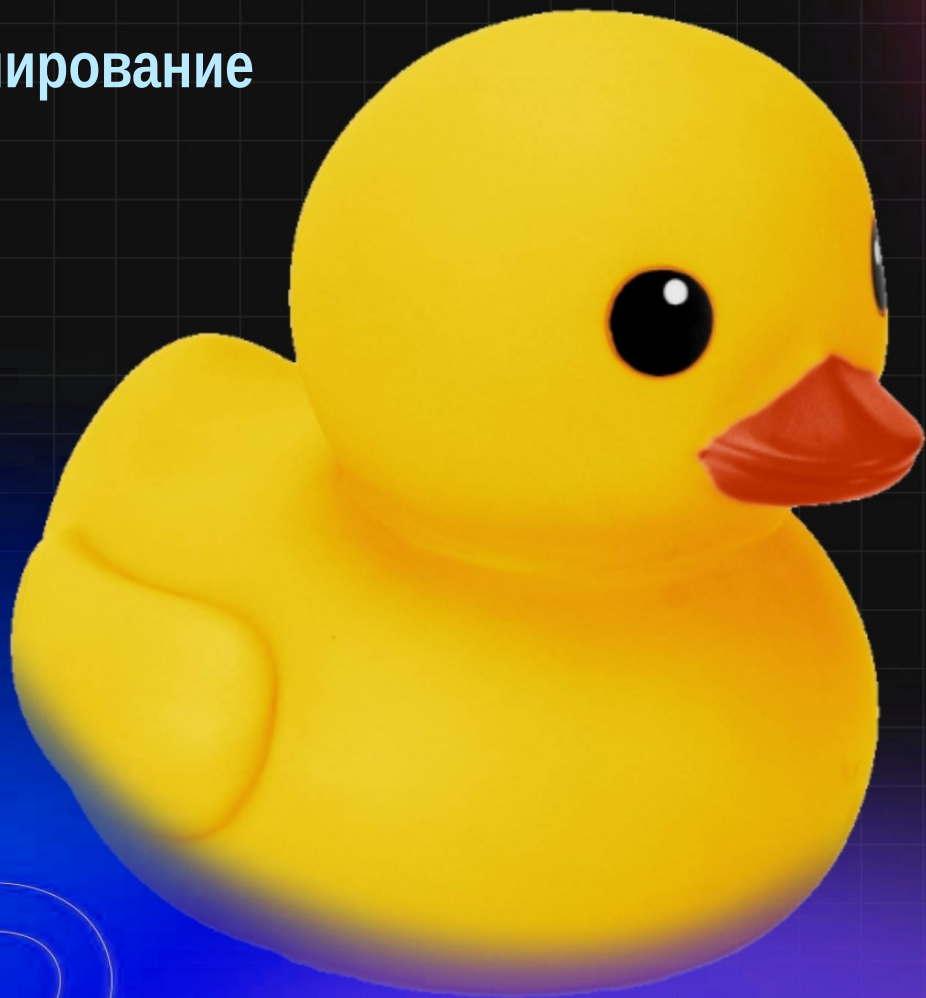
- PipedOutputStream
- PipedWriter
- PipedInputStream
- PipedReader

Шаблоны проектирования

pos.connect(pis)



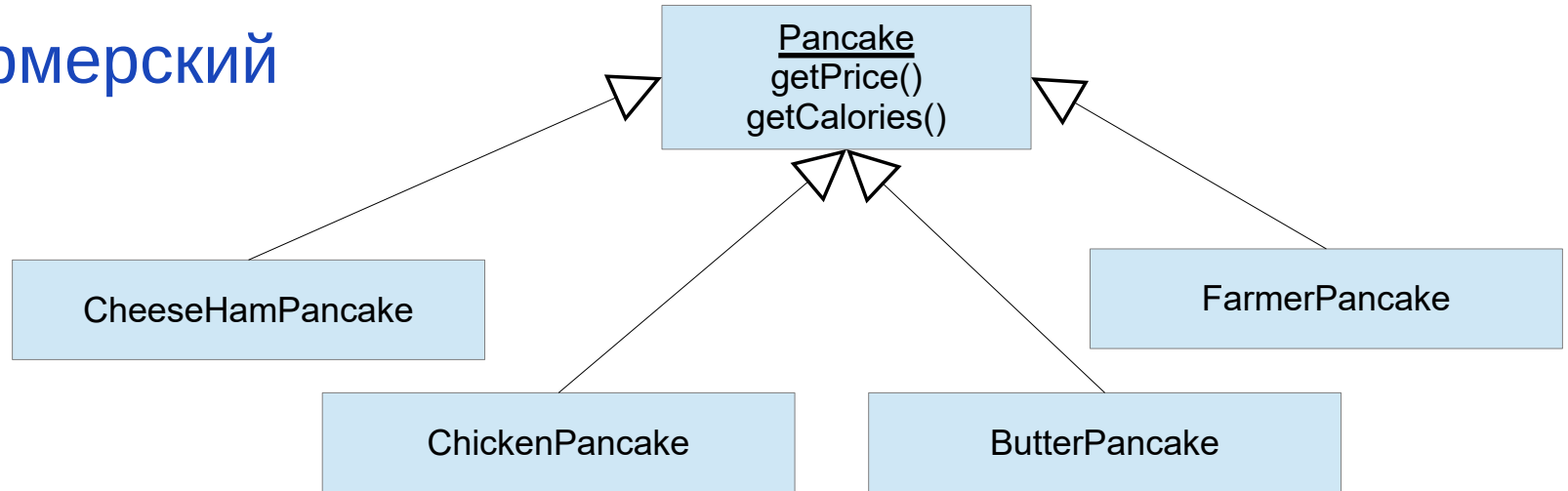
Программирование
2 семестр
2025



ІТМО

(продолжение)

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



- Блин с ветчиной и сыром
- Блин с куриной грудкой
- Блин с маслом
- Блин Фермерский
- Добавки
 - ❖ Огурцы
 - ❖ Лук-фри
 - ❖ Картофельное пюре
 - ❖ Сметана
- `ChickenPancakeWithPicklesAndMashedPotatoes`
- `FarmerPancakeWithSourCreamAndOnionAndPickles`

- Блин с ветчиной и сыром
- Блин с куриной грудкой
- Блин с маслом
- Блин Фермерский
- Добавки
 - ❖ Огурцы 30 руб.
 - ❖ Лук-фри
 - ❖ Картофельное пюре
 - ❖ Сметана

```
class PicklePancake extends Pancake {  
    Pancake base;  
    PicklePancake(Pancake p) { base = p; }  
    double getPrice() { return base.getPrice() + 30; }  
}
```

Шаблон Decorator - Теремок

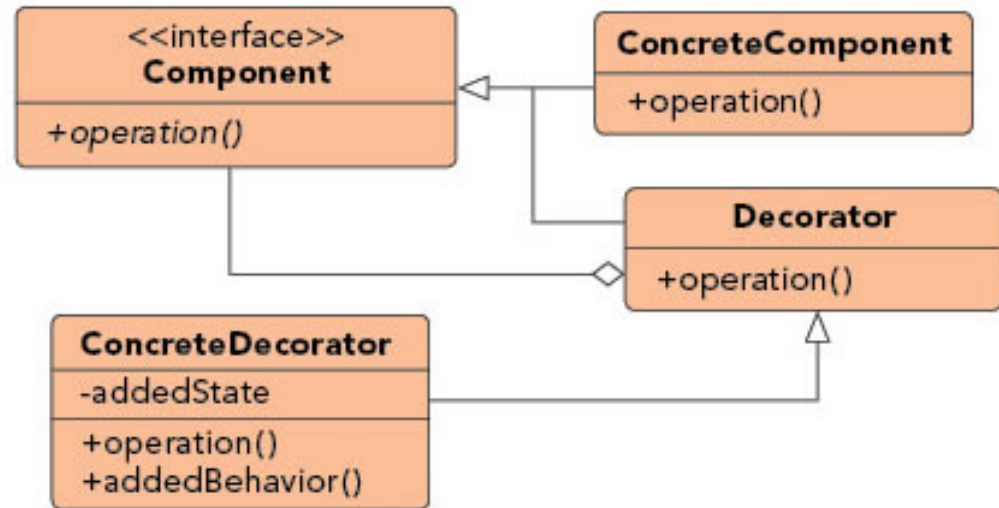
- Блин с ветчиной и сыром
- Блин с куриной грудкой
- Блин с маслом
- Блин Фермерский
- Огурцы
- Лук-фри
- Картофельное пюре
- Сметана

```
Pancake p =  
    new MashedPotatoPancake(  
        new PicklePancake(  
            new CheeseHamPancake()));  
  
p.getPrice();
```

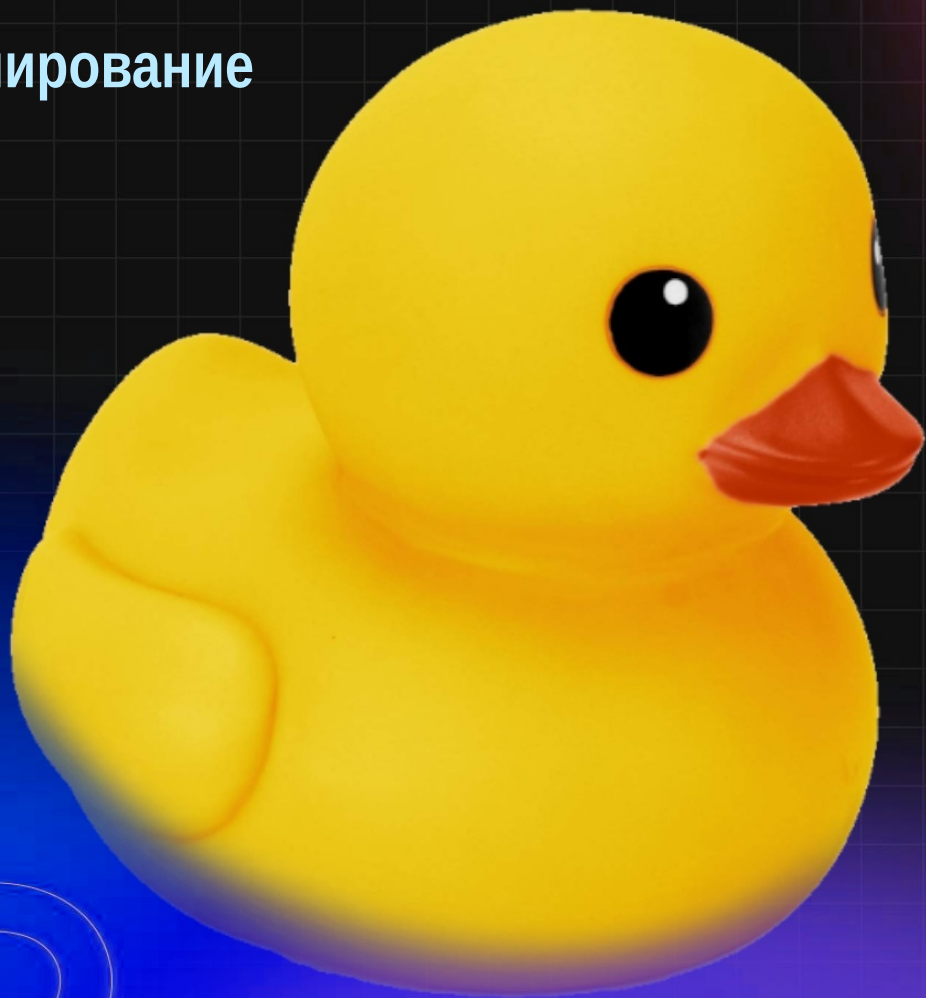


- Позволяет добавлять функциональность динамически
- Вместо большой иерархии - несколько декораторов
- Сложность конфигурирования

В



Программирование
2 семестр
2025



ІТМО

(продолжение)

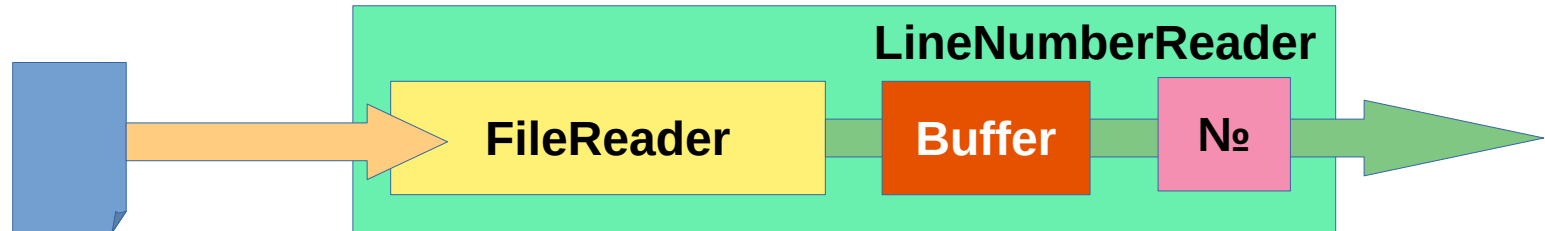
- `FilterInputStream(InputStream)`
- `FilterOutputStream(OutputStream)`
- `FilterReader(Reader)`
- `FilterWriter(Writer)`
 - ❖ исходный поток = аргумент конструктора
 - ❖ поток-фильтр = декоратор



- `BufferedInputStream`
- `BufferedReader`
- Бuffer для повышения производительности
- Возможность построчной работы
- `BufferedOutputStream`
- `BufferedWriter`



- `LineNumberInputStream`
- `LineNumberReader` extends `BufferedReader`
 - ❖ `getLineNumber()`
 - ❖ `setLineNumber(int)` - меняет только номер, не саму строку



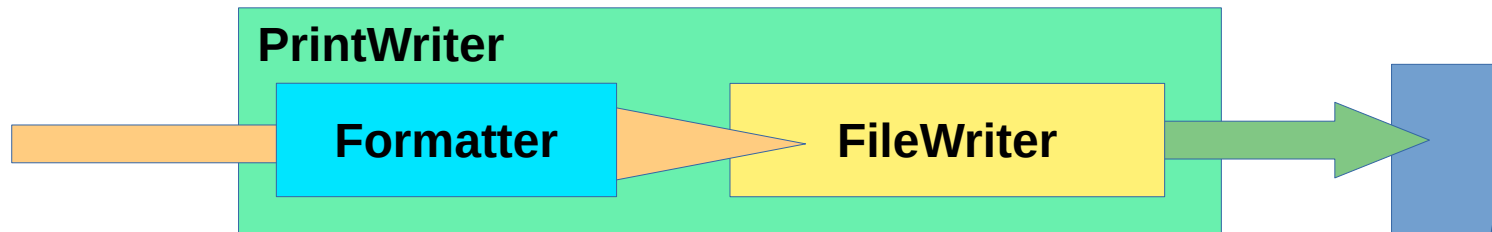
```
try (LineNumberReader in = new LineNumberReader(  
    new FileReader("in.txt"));  
    BufferedWriter out = new BufferedWriter(  
        new FileWriter("out.txt")) {  
    String line;  
    while ((line = in.readLine()) != null) {  
        out.write(in.getLineNumber() + ": " + line);  
        out.newLine();  
    }  
    out.flush();  
} catch (IOException e) {  
    System.err.println(e);  
}
```

- Чтение-запись в других кодировках
- **InputStreamReader** extends Reader
 - ❖ байты в символы
 - ❖ `new InputStreamReader(InputStream in, кодировка)`
- **OutputStreamWriter** extends Writer
 - ❖ символы в байты
 - ❖ `new OutputStreamWriter(OutputStream out, кодировка)`

- `PrintStream, PrintWriter`
- `print, println` — один аргумент
- `printf, format` — форматная строка + аргументы

```
format("%c = %2$+9.7f", 'π', Math.PI);
```

```
π = +3,1415927
```



• %[индекс\$][флаги][размер][.точность]формат

• %b - boolean

• %d - decimal

• %f - float

• %h - hashCode

• %o - octal

• %e - exponent

• %t = time/date

• %h - hex

• %g - %e / %f

• %% - %

• %s - string

• %a - hex float

• %n - newline

• %c - char

• индекс — номер аргумента

• размер — количество СИМВОЛОВ

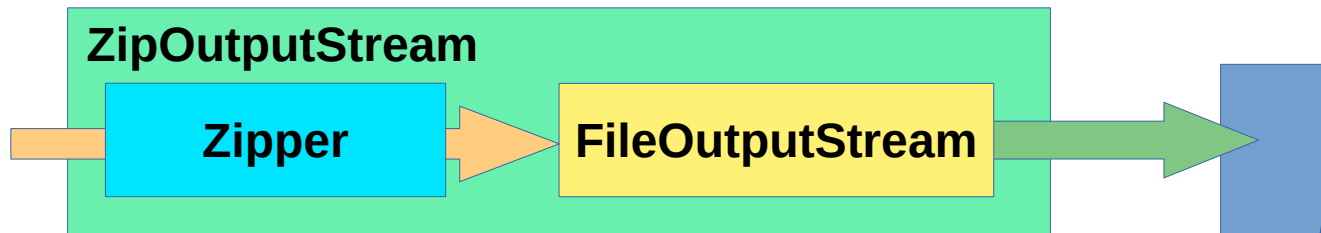
• флаги — зависят от формата

• точность — после запятой

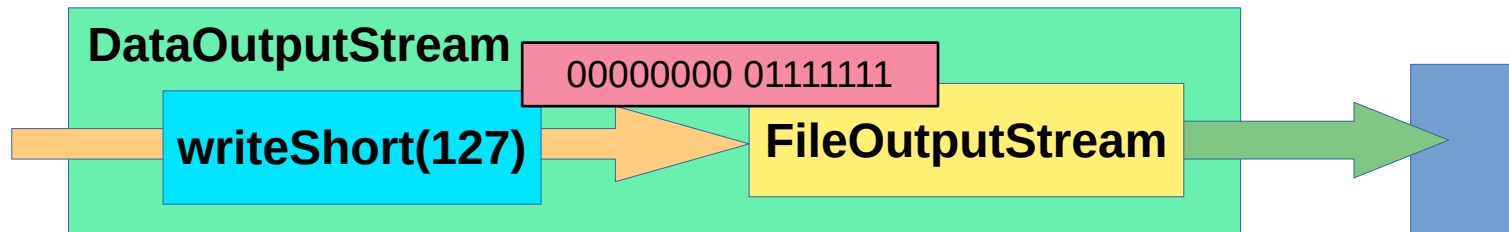
```
format("%c = %2$+9.7f", 'π', Math.PI);
```

```
π = +3,1415927
```

- потоки с компрессией
- InflaterInputStream
 - ❖ GzipInputStream
 - ❖ ZipInputStream
 - JarInputStream
- DeflaterOutputStream
 - ❖ GzipOutputStream
 - ❖ ZipOutputStream
 - JarOutputStream



- DataInputStream, DataOutputStream
- примитивы и строки ↔ байты
- Последовательность при чтении - такая же как при записи:
- DataOutputStream.writeInt(int), DataInputStream.readInt()
 - byte, short, long, float, double, char, boolean
 - writeUTF(String), readUTF()



- Сериализация — запись объектов в виде потока байтов
- Классы ObjectOutputStream, ObjectInputStream
- Интерфейс-метка Serializable
- При записи объекты записываются с порядковым номером (serial number)
- Объекты записываются в поток иерархически (deep copy)
- Объект записывается в поток только один раз, потом используется ссылка на номер объекта
- При чтении одного и того же объекта из одного потока, он восстанавливается один раз
- При чтении объекта из двух потоков, объект восстанавливается дважды

- Те же методы, что и у DataOutputStream, DataInputStream
 - ❖ + writeObject(Object)
 - ❖ + Object readObject()
- Методы записывают/читают:
 - ❖ класс объекта
 - ❖ сигнатуру класса (зависит от имени и модификаторов класса, интерфейсов, типов и имен полей, имен и сигнатур конструкторов и методов)
 - ❖ сигнатура класса может быть записана в поле serialVersionUID
 - ❖ значения нестатических и непереходных полей данного класса, и всех его суперклассов
 - ❖ **не сериализуются** поля с модификаторами static и transient

- Сериализованный объект зависит от внутренней структуры класса
- Объекты могут создаваться в обход конструкторов
- Потенциальные проблемы совместимости версий
- Возможные проблемы с:
 - ❖ продолжительностью сериализации
 - ❖ необходимой памятью на сериализацию
 - ❖ необходимой глубиной стека

- Задать несериализуемым полям модификатор `transient`
- Реализовать методы в сериализуемом классе
 - ❖ `private void writeObject(ObjectOutputStream os)`
 - ❖ `private void readObject(ObjectInputStream is)`
- ВНУТРИ ЭТИХ МЕТОДОВ ВЫЗЫВАЮТСЯ МЕТОДЫ
 - ❖ `os.defaultWriteObject()`
 - ❖ `is.defaultReadObject()`
- Суперкласс не трогаем

- Реализуем интерфейс Externalizable
- реализуем методы (полный контроль сериализации)
 - ❖ writeExternal(ObjectOutput o)
 - ❖ readExternal(ObjectInput i)
- Необходимо самостоятельно обрабатывать суперкласс
- При сериализации вместо стандартного механизма будет вызван метод writeExternal, при десериализации - readExternal

- Стандартные потоки ввода-вывода
 - ❖ `InputStream System.in`
 - ❖ `PrintStream System.out`
 - ❖ `PrintStream System.err`
- `java.io.Console`
 - ❖ `Console c = System.console()`
 - ❖ `cons.readLine()`
 - ❖ `cons.readPassword()`
 - ❖ `cons.printf()`
 - ❖ `cons.format()`

- Конструкторы

- ❖ Scanner(File)
- ❖ Scanner(Path)
- ❖ Scanner(InputStream)
- ❖ Scanner(Readable)
- ❖ Scanner(String)

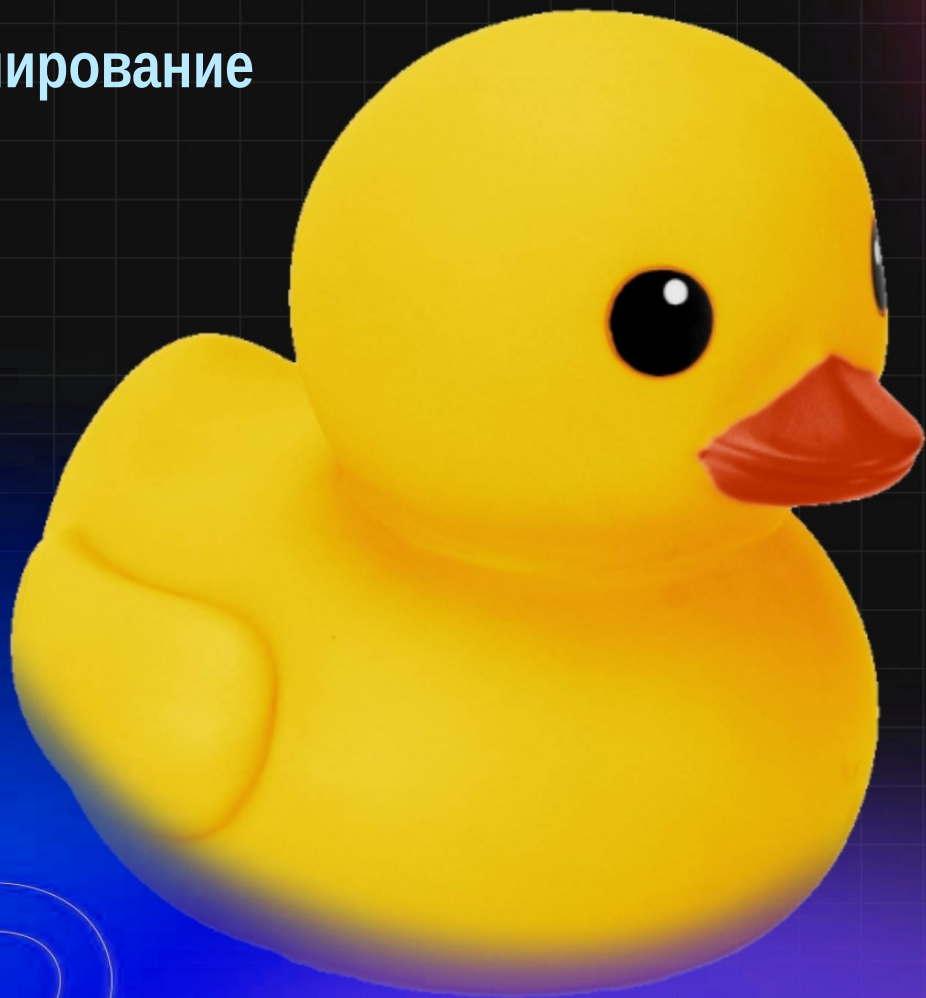
- Методы

- ❖ boolean hasNext(), String next()
- ❖ boolean hasNextInt(), nextInt()
 -long, byte, short, float, double, boolean, char
- ❖ String nextLine(), nextPattern()
- ❖ void useDelimiter(String)
- ❖ void useRadix(int)

```
Scanner scanner = new Scanner(System.in);
List<List<Integer>> ints = new ArrayList<>();
List<Integer> intList = null;
while (scanner.hasNext()) {
    String line = scanner.nextLine();
    Scanner intScanner = new Scanner(line);
    intList = new ArrayList<Integer>();
    while (intScanner.hasNextInt()) {
        intList.add(intScanner.nextInt());
    }
    ints.add(intList);
    intScanner.close();
}
```

```
100 200 300 400 500
1 2 3 4 5 6 7 8 9
-1 -2 -3 -4 -5
9 8 7 6 5 4 3 2 1
45 0 -45
```

Программирование
2 семестр
2025

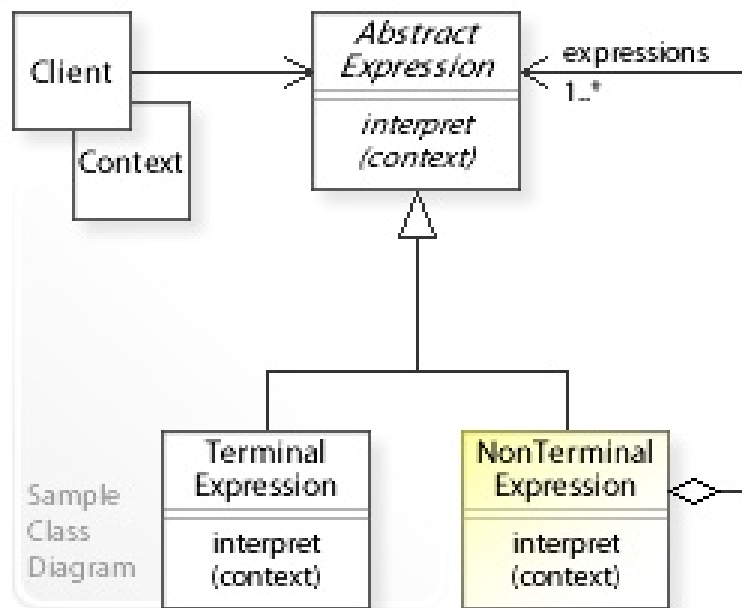
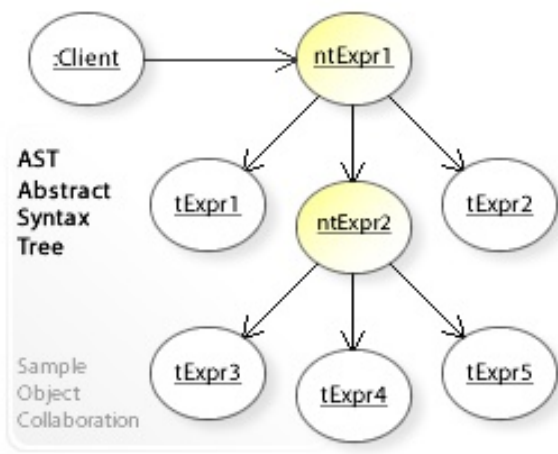


ІТМО

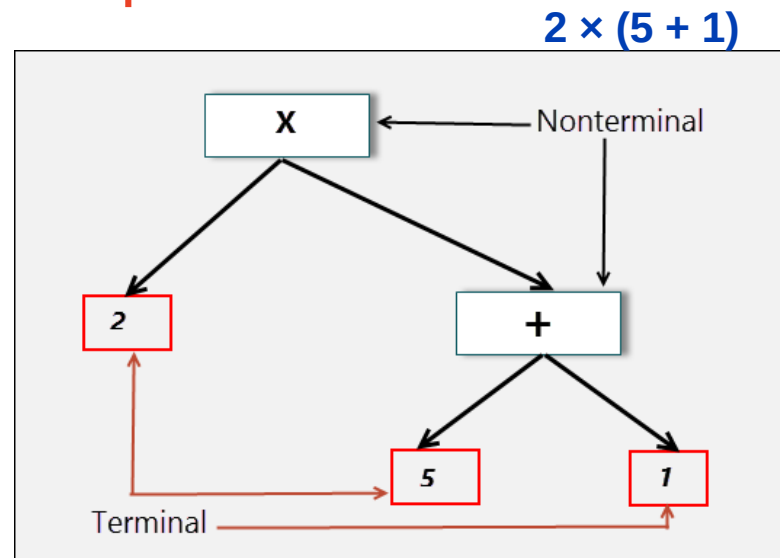
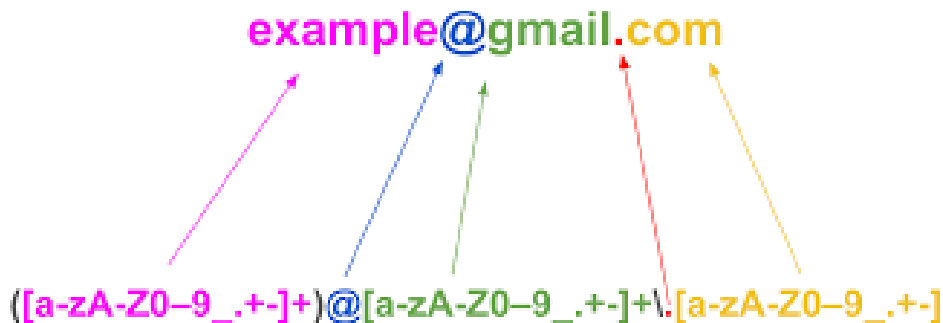
**Вспомогательные
классы**

- `public Random()`
- `public Random(long seed)`
- `nextInt()`, `nextDouble`, `nextLong`, `nextGaussian`
- `IntStream ints()`
- `LongStream longs()`
- `DoubleStream doubles()`

- Интерпретатор языка для управления поведением
 - ❖ Регулярные выражения
 - ❖ Форматирование строк



- Простота расширения и изменения языка
- Простота добавления новых способов
- Сложность сопровождения сложных грамматик



- Класс Pattern — представляет регулярное выражение
- Класс Matcher — движок, проверяющий соответствие

```
String regex = "a*b";
```

```
Pattern p = Pattern.compile(regex);
```

```
Matcher m = p.matcher("aaabbbb");
```

```
boolean b = m.matches();
```

```
boolean b = Pattern.matches(regex, "aaabbbb");
```

`x - x`

`\\ - \`

`\0nnn` - 8-ричный код (байт)

`\xhh` - 16-ричный код (байт)

`\uhhhh` - 16-ричный код (Unicode)

`\t` - tab

`\n` - newline

`\r` - carriage-return

подстрока символов

in

начало и конец строки

^lo

ua\$

lorem ipsum dolor sit
amet, consectetur
adipisc**in**g elit, sed
do eiusmod tempor
incididunt ut labore
et dolore magna
aliqu**ua**

СИМВОЛЬНЫЙ КЛАСС

[bp]or
i[^dtns]
e[l-n]

lorem ipsum dolor sit
amet, consectetur
adipiscing elit, sed
do eiusmod tempor
incididunt ut labore
et dolore magna
aliqua

символьный класс -
сокращенное обозначение

`\w\w\wt\W`

`\d` цифра

`\D` не цифра

`\w` буква

`\W` не буква

`\s` пробел

`\S` не пробел

lorem ipsum dolor sit
amet, consectetur
adipiscing elit, sed
do eiusmod tempor
incididunt ut labore
et dolore magna
aliqua

любой символ

`...C`

lorem ipsum dolor sit
amet, consectetur
adipiscing elit, sed
do eiusmod tempor_
incididunt ut labore
et dolore magna
aliqua

альтернатива

`s(i|ec)t`

lorem ipsum dolor sit
amet, consectetur
adipiscing elit, sed
do eiusmod tempor
incididunt ut labore
et dolore magna
aliqua

квантификаторы

\s.{6}\s

e.{0,2}i

s\S{0,1}m

loremipsum dolor sit
amet, consectetur
adipiscing elit, sed
do eiusmod tempor
incididunt ut labore
et dolore magna
aliqua

квантификаторы

$s \underline{S}^? m$ $\{0, 1\}$

$b \underline{.}^+ d$ $\{1, \}$

$e t \underline{[^\wedge o]}^*,$ $\{0, \}$

lorem ipsum dolor sit
amet, consectetur
adipiscing elit, sed
do eiusmod tempor
incididunt ut labore
et dolore magna
aliqua

жадность

lor.*it

\si\w*i

lorem ipsum dolor sit
amet, consectetur
adipiscing elit, sed
do eiusmod tempor
incididunt ut labore
et dolore magna
aliqua

группы

```
(\w+).*\1,  
1 = it
```

lorem ipsum dolor sit
amet, consectetur
adipiscing elit, sed
do eiusmod tempor
incididunt ut labore
et dolore magna
aliqua

```
\w+@(\w+\. )+\w{2,}
```

```
user@se.itmo.ru
```

группы

$(\backslash w) \dots \backslash 1 (\dots) \{ 1, 6 \} \backslash 2$

1 = i 2 = d

1 = t 2 = 

lorem ipsum dolor sit
amet, consectetur
adipiscing elit, sed
do eiusmod tempor
incididunt ut labore
et dolore magna
aliqua

- Запись
 - `println()` / `format()`
- Чтение
 - `Scanner`, `StreamTokenizer`, `Regex`
- CSV
 - ! специальные символы, разделители, переносы строк

- Binding
 - ❖ XML/JSON <-> Java Object (целиком)
 - ❖ низкая скорость, настройка на класс, много памяти, простая обработка
 - ❖ JAXB
- DOM (Document Object Model)
 - ❖ XML/JSON <-> Object Tree (Graph)
 - ❖ Node, Element
 - ❖ универсальная модель, много памяти, средняя сложность
 - ❖ XML DOM
- Event/Stream
 - ❖ element start / element end
 - ❖ высокая скорость, мало памяти, сложнее обработка
 - ❖ SAX, StAX

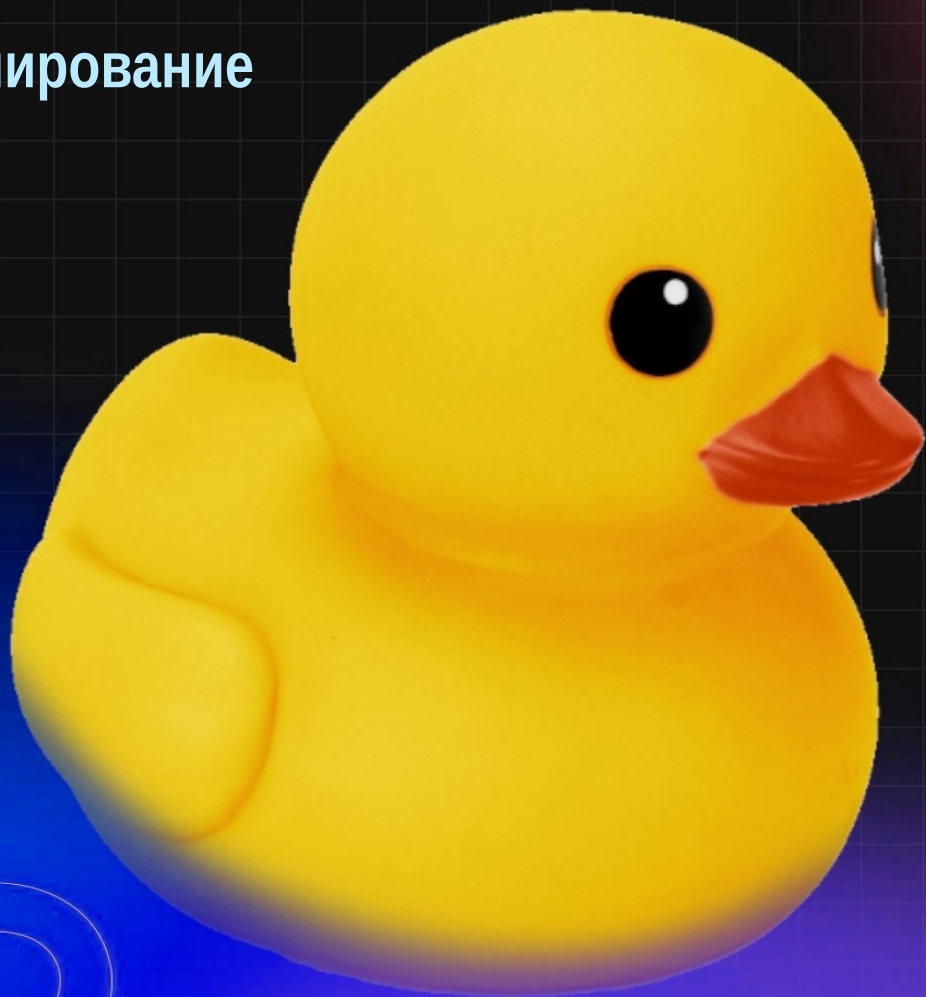
- Стандартные библиотеки java
 - XML
 - модуль `java.xml` - переместился в Java EE
 - JSON
 - `javax.json.*` (Java EE)

- Сторонние библиотеки
 - ❖ CSV
 - Apache Commons CSV
 - OpenCSV
 - ❖ JSON
 - GSON
 - Jackson
 - ❖ XML
 - JacksonXML
 - Jakarta XML Bind

```
/**
 * This is my class
 * @author I
 */
public class MyClass {
    /**
     * This is my method
     * @params x just x
     * @return double x
     */
    public int doubleX(int x) {
        return x * 2;
    }
}
```

```
javadoc -d doc *.java
```

Программирование
2 семестр
2025



ІТМО

Файлы (IO / NIO.2)

- Файловая система
 - ❖ Каталоги
 - ❖ Файлы
 - Жесткие ссылки
 - Символические ссылки
- Путь к файлу
 - ❖ абсолютный (от корня)
 - ❖ относительный
- Windows (NTFS)
 - ❖ отдельные FS
 - ❖ разделитель - \
 - ❖ C:\Users\student\file.txt
- Linux / UNIX / MacOS
 - ❖ виртуальная FS
 - ❖ разделитель - /
 - ❖ /home/student/file.txt

- Класс **java.io.File** - все операции с путями и файлами
- Интерфейс **java.nio.file.Path**
 - ❖ символические ссылки и расширенные атрибуты
 - ❖ альтернативные файловые системы
 - ❖ копирование и перемещение файлов
 - ❖ методы бросают исключения
 - ❖ настраиваемый разделитель
- Преобразование между File и Path
 - ❖ `File PathToFile()`
 - ❖ `Path File.toPath()`

- abstract class `java.nio.file.FileSystem` — файловая система

- ❖ Методы:

- `Iterable<Path> getRootDirectories()`

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

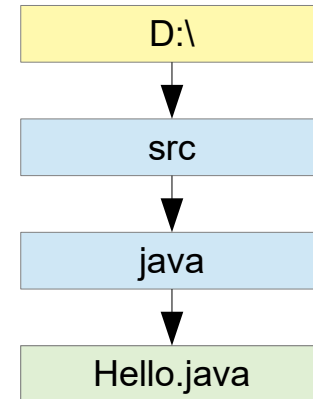
- `String getSeparator()`

- класс-утилита `java.nio.file.FileSystems`

- `FileSystem getDefault()`

- интерфейс `java.nio.file.Path` — путь к файлу
 - ❖ абсолютный / относительный
 - ❖ пустой — текущая директория
- класс-утилита `java.nio.file.Paths`

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



- Класс-утилита `java.nio.file.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)
```

```
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