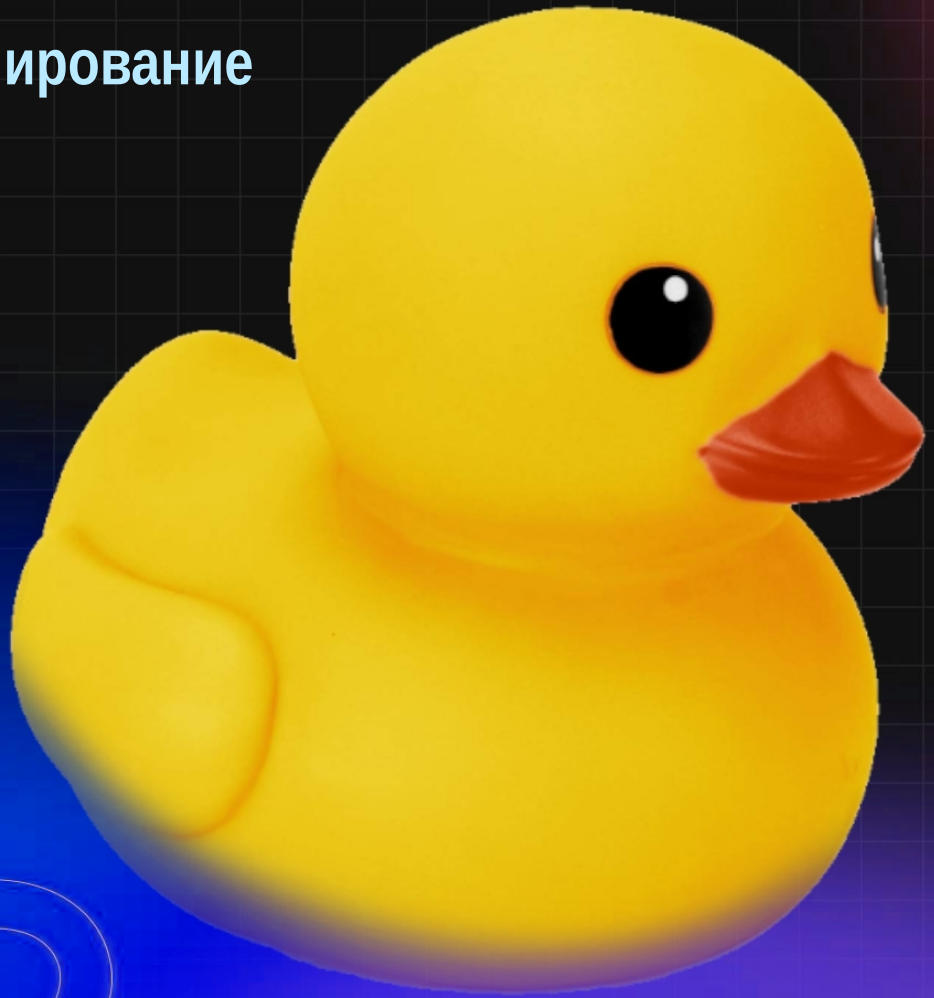
Программирование
2 семестр
2024



ИТМО

Сравнение элементов

Сравнение элементов

ИТМО

- 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

Обобщенный алгоритм сортировки

ИТМО

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

Сортировка (естественная)

ИТМО

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

class String implements Comparable
```

Java

Сортировка (отдельный компаратор)

ИТМО

```
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)

ИТМО

```
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)

ИТМО

```
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)

ИТМО

```
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)

ИТМО

```
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); }
  }
}
```

Java

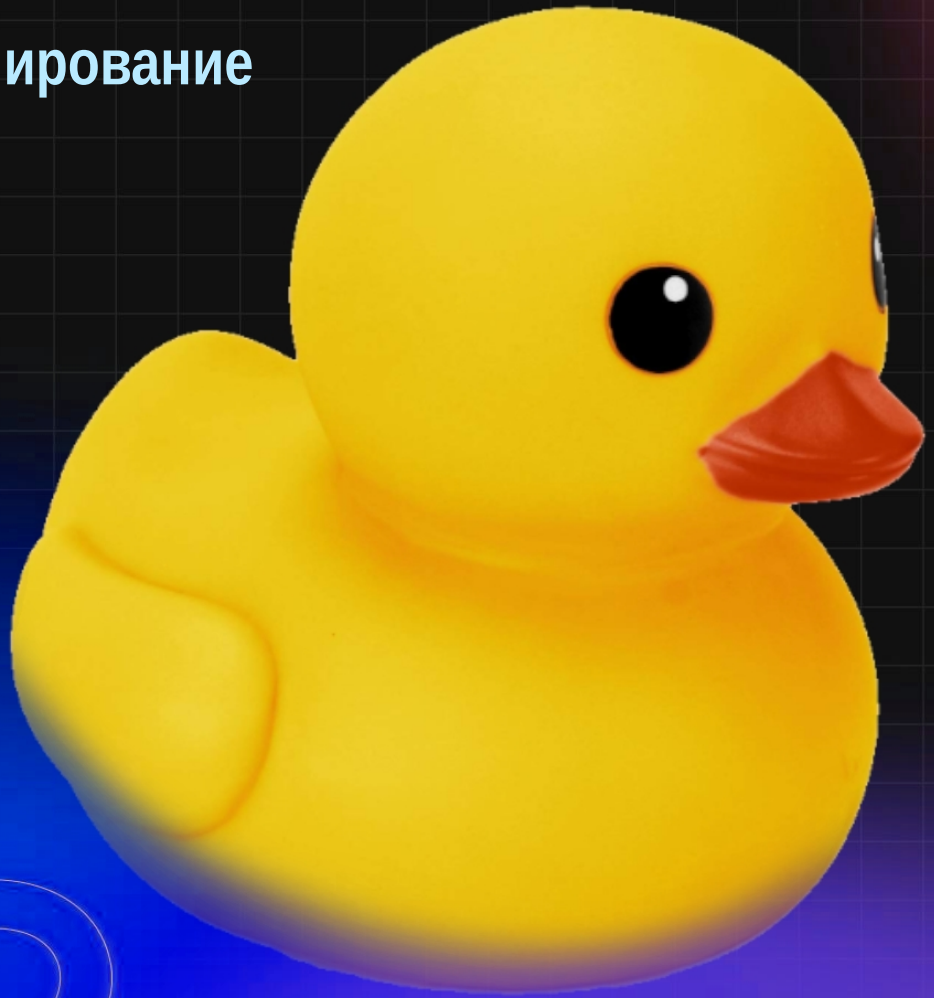
Сортировка (лямбда-выражение)

ИТМО

```
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); }
  }
}
```

Java

Программирование
2 семестр
2024



ИТМО

Ввод-вывод

Ввод-вывод в Java

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

Ввод-вывод

Устройство ввода

драйвер ОС

API

Программа

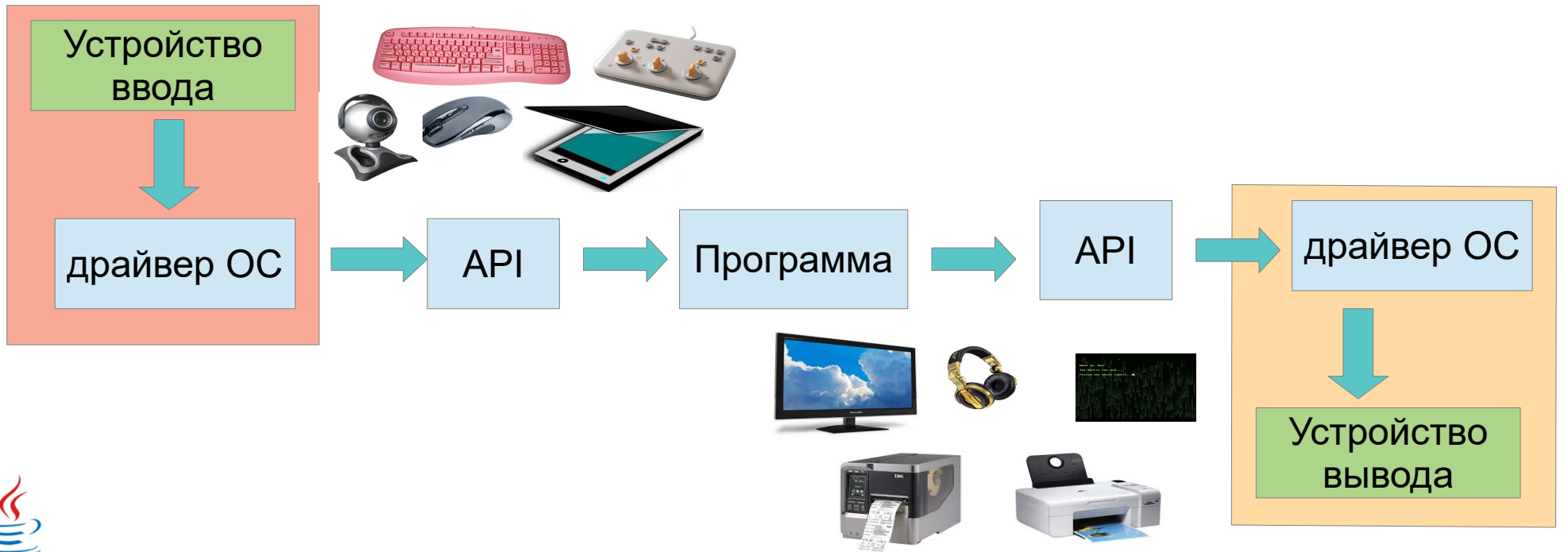
API

драйвер ОС

Устройство вывода

Абстракция ввода-вывода

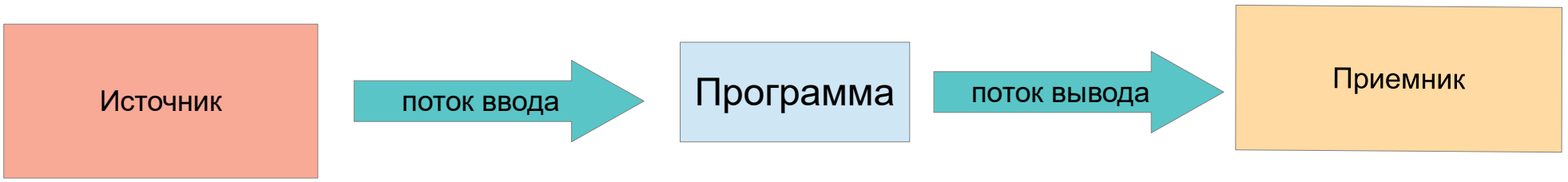
- Потоки ввода-вывода
- Источник данных и приемник данных



Абстракция ввода-вывода

ИТМО

- Потоки ввода-вывода
- Источник данных и приемник данных



Java

Пакет java.io

ИТМО

- 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

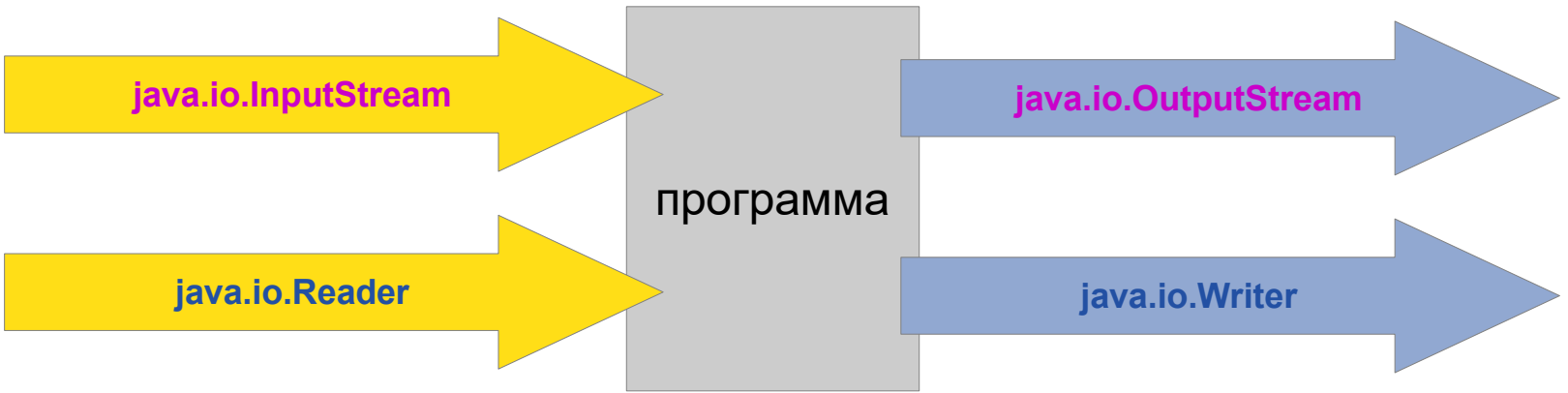
Java

Байты и символы

```
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'
```

Потоки ввода-вывода

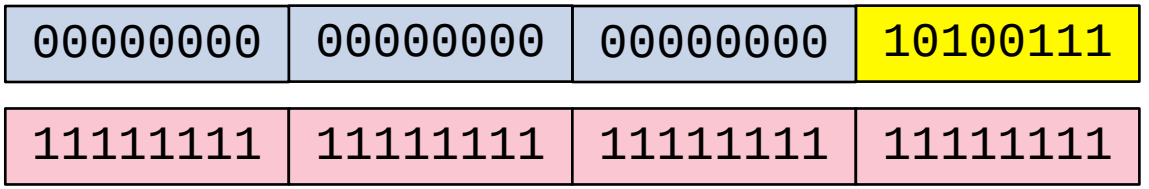
- Базовые абстрактные классы для потоков
- Ввода и вывода
- Байтовые и символьные



java.io.InputStream

ИТМО

- abstract **int** read()
  - прочитанный байт
  - -1 (конец потока)



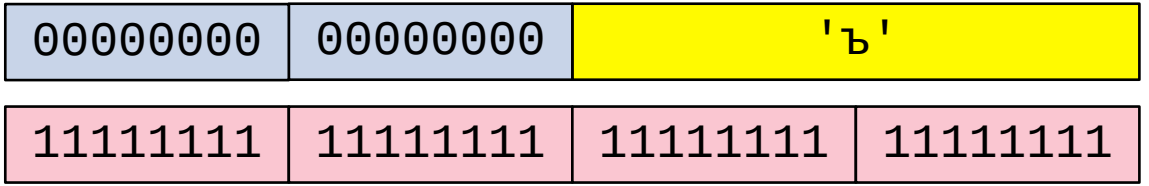
- int read(byte[] buf, int off, int len)

- int available()
- long skip(long n)
- void close()
- boolean markSupported()
- void mark(int limit)
- void reset()

java.io.Reader

ИТМО

- 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()

java.io.OutputStream

ИТМО

- abstract void write(**int** b)
  - записываемый байт

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

- void flush()
- void close()

java.io.Writer

- 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)

Interface Flushable

ИТМО

- 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

Java

Интерфейс Closeable

ИТМО

- close() throws IOException - освобождение ресурса

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

Java

Интерфейс AutoCloseable

ИТМО

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

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

Java 9

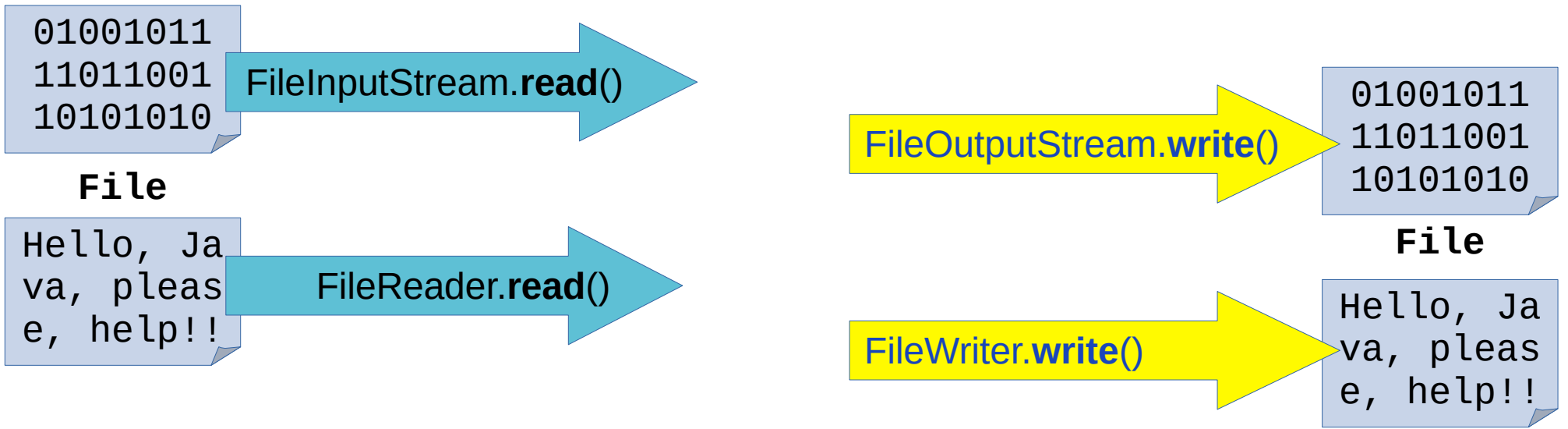
```
InputStream ins = new InputStream();
try(ins) {
    ins.read();
}
```

Java

Специализированные потоки - File

- FileOutputStream
- FileWriter

- FileInputStream
- FileReader



Чтение и запись с помощью потоков

ИТМО

Java 9

```
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);
}
```

Java

Чтение и запись с помощью потоков

ИТМО

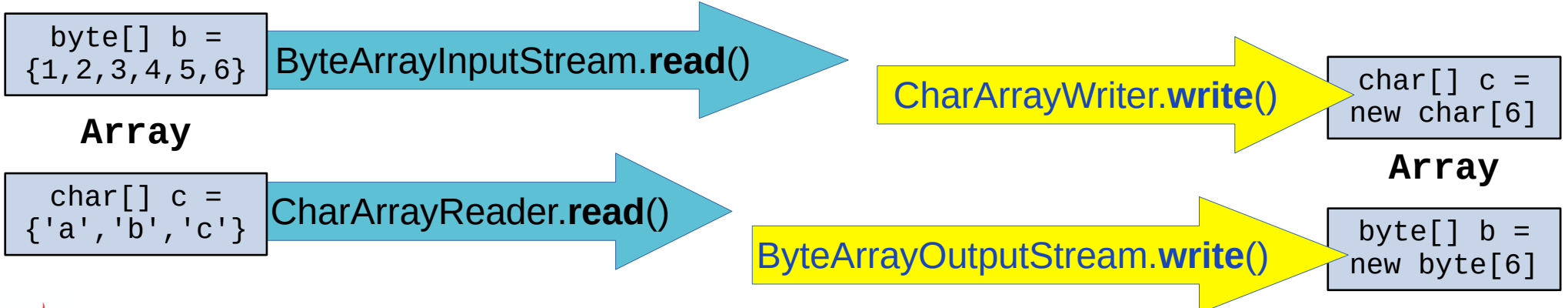
```
try (FileReader in = new FileReader("in.txt");
     FileWriter out = new FileWriter("out.txt")) {
  int c;
  while ((c = in.read()) != -1) {
    out.write(c);
  }
  out.flush();
} catch (IOException e) {
  System.err.println(e);
}
```

Java

Специализированные потоки - Array

ИТМО

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

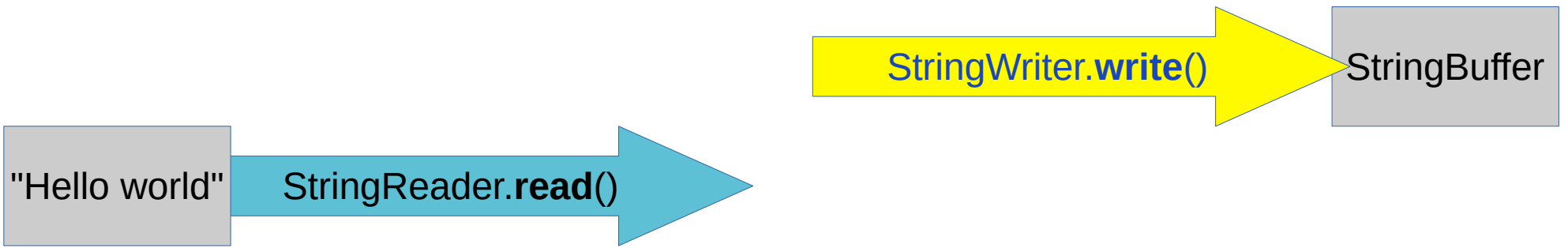


Java

Специализированные потоки - String

ИТМО

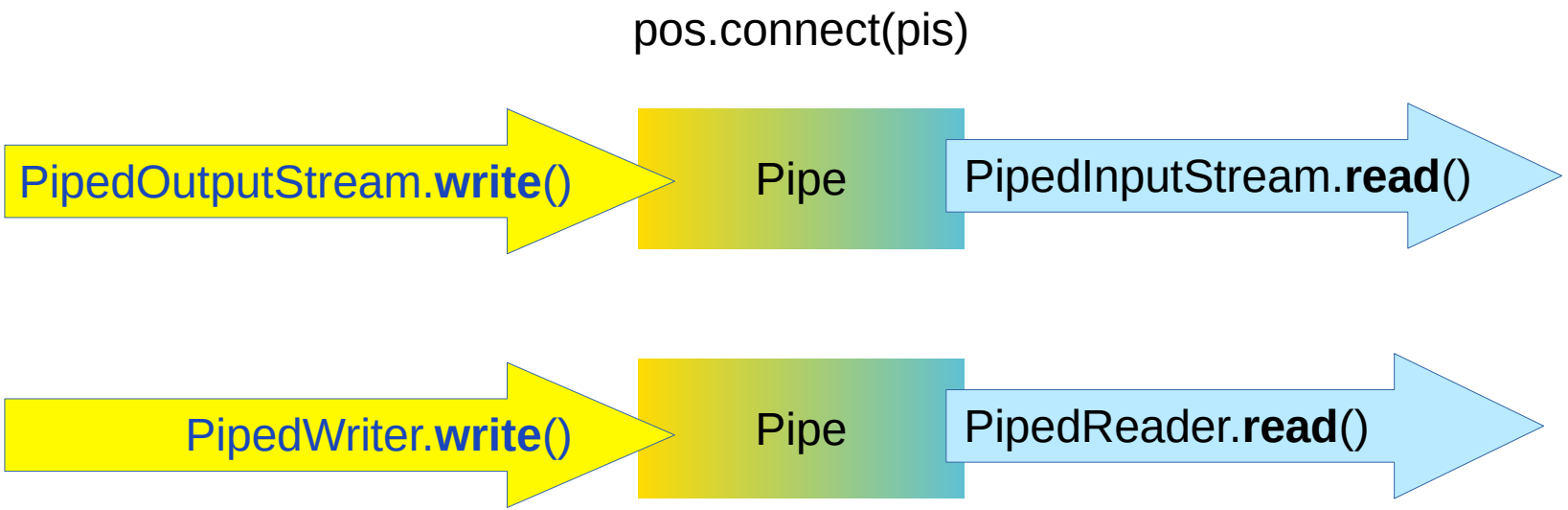
- StringWriter
  - StringBuffer getBuffer()
  - String toString()
- StringReader



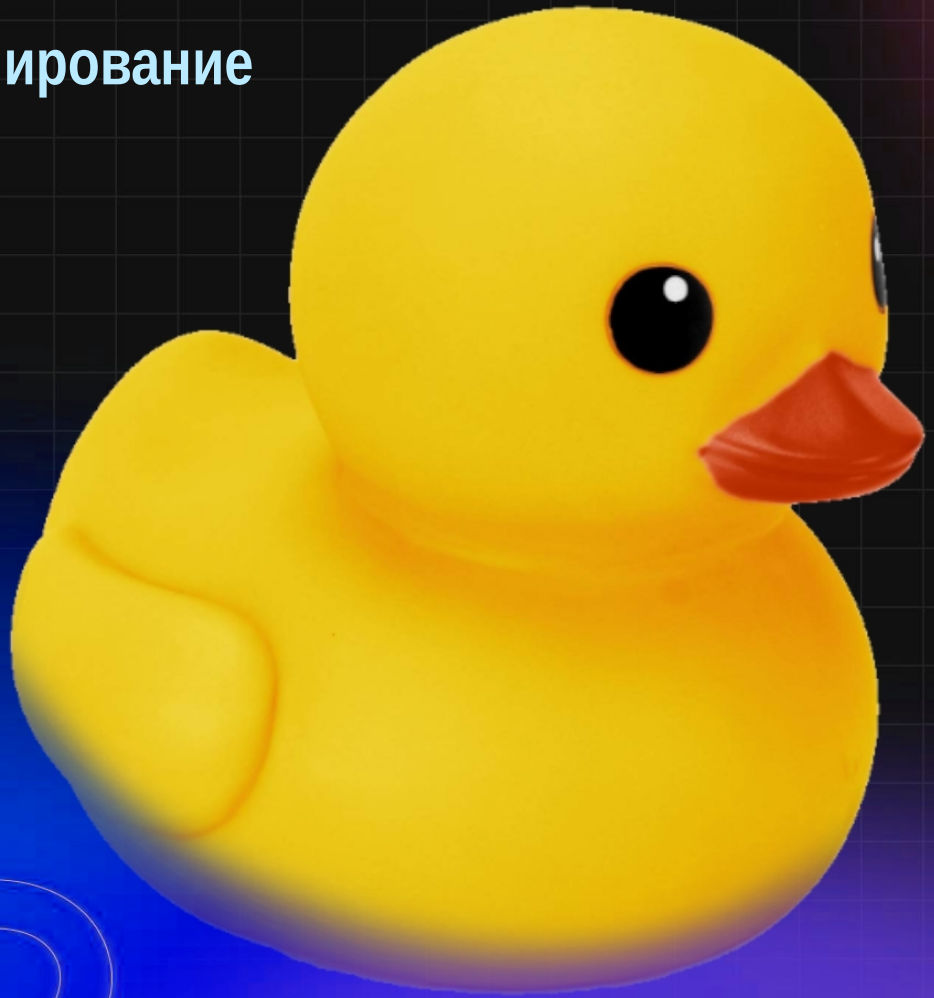
Java

Специализированные потоки - Pipe

- PipedOutputStream
- PipedWriter
- PipedInputStream
- PipedReader



Программирование
2 семестр
2024



ИТМО

Шаблоны проектирования (введение)

Шаблоны проектирования

- Кто-то когда-то уже делал что-то похожее
- Пришлось вносить изменения - возникли проблемы
- Нужно сразу было делать по-другому!

- GoF Book
  - Gang of Four

Основные принципы проектирования

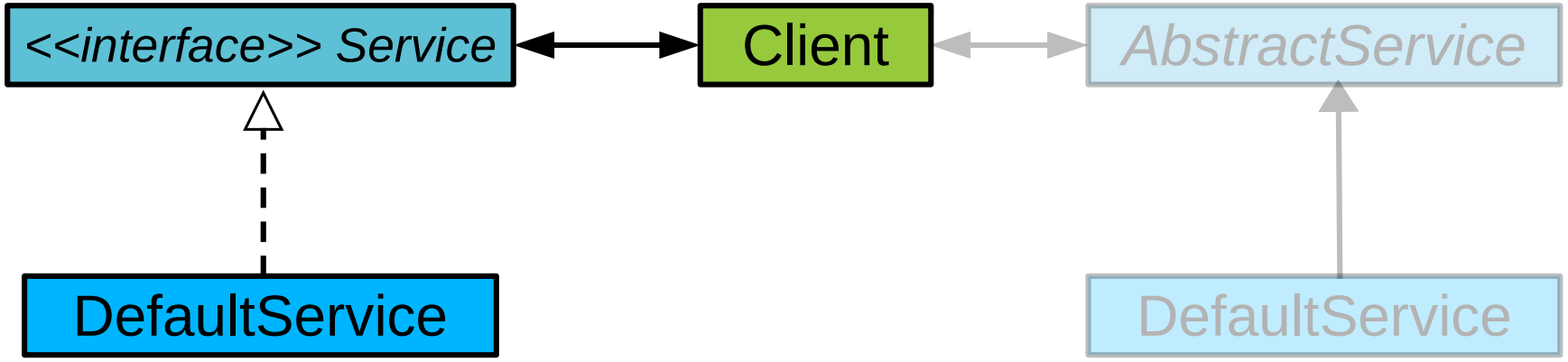
ИТМО

- Повторное использование кода
  - DRY - Don't repeat yourself
  - стандартные библиотеки
  - фреймворки
- Расширяемость
  - Учет возможных будущих изменений

Java

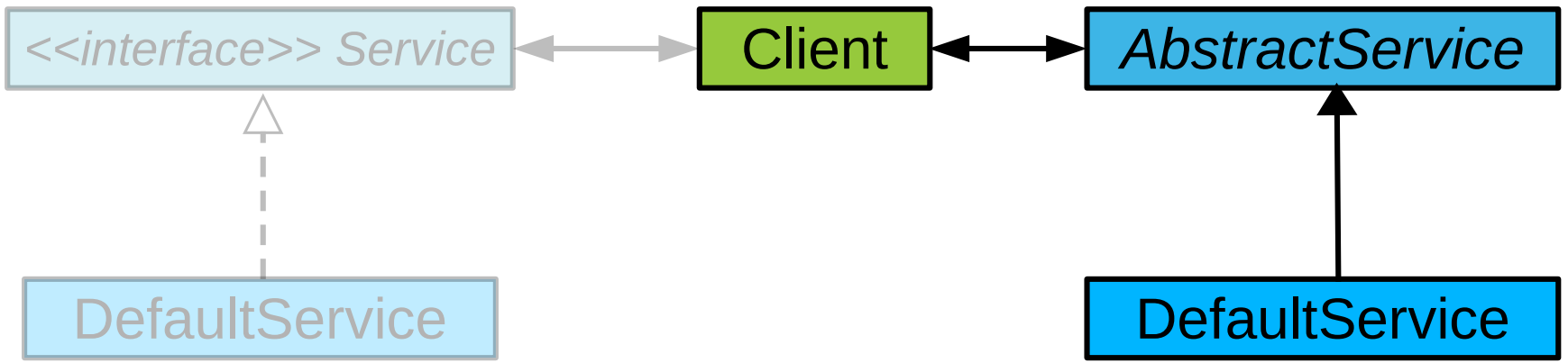
Интерфейс / Абстрактный суперкласс

- Независимость от реализации - больше гибкость и универсальность
- Унификация поведения потомков - проще реализация



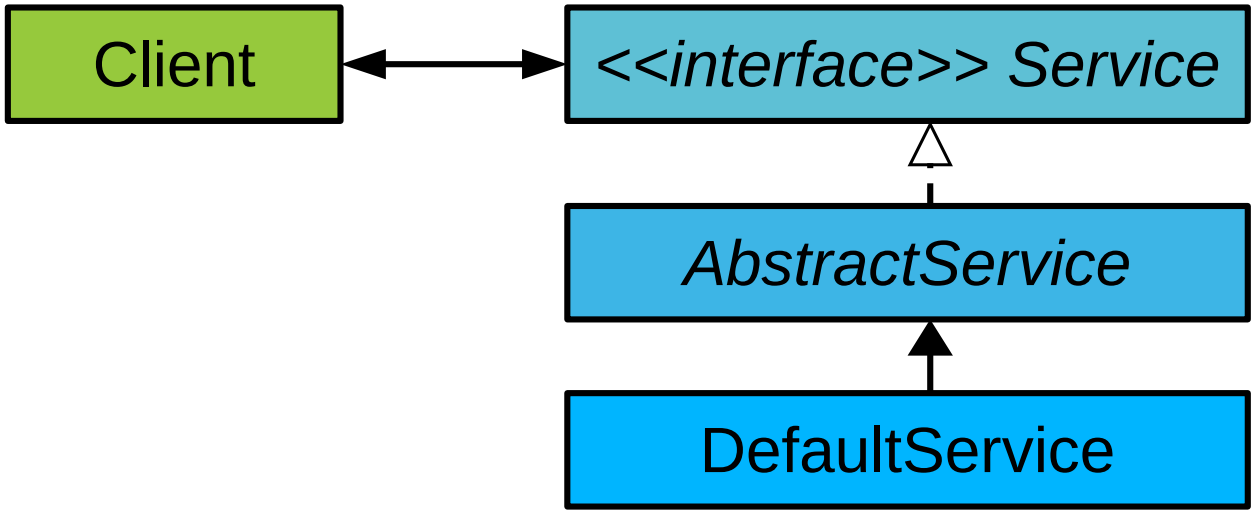
Интерфейс / Абстрактный суперкласс

- Независимость от реализации - больше гибкость и универсальность
- Унификация поведения потомков - проще реализация



Интерфейс + абстрактный суперкласс

- Общие свойства - АБСТРАКТНЫЙ КЛАСС
- Взаимодействие с абстракцией - ИНТЕРФЕЙС
- Set<T> ⇠ AbstractSet<T> ⇦ HashSet<T>



Наследование и делегирование

ИТМО

- Наследование - подкласс
  - собака - подкласс животного
  - статическое отношение **is-a** (Dog is an animal)

```
class Dog extends Animal  { }
```

Java

Наследование и делегирование

ИТМО

- Наследование - подкласс
  - собака - подкласс животного
  - статическое отношение **is-a** (Dog is an animal)
- Student и Person
  - студент - **роль**

```
class Dog extends Animal  { }

class Person { }

class Student extends Person { }
class Teacher extends Person { }
```

Java

Наследование и делегирование

ИТМО

- Наследование - подкласс
  - собака - подкласс животного
  - статическое отношение **is-a** (Dog is an animal)
- Student и Person
  - студент - **роль**
  - роль может меняться
- Делегирование!

```
class Dog extends Animal  { }

class Person {
  public getName() { }
}

class Student {
  Person p;
  public getName() {
    return p.getName();
  }
}
```

Java

Наследование и композиция

ИТМО

- Наследование - подкласс
  - собака - подкласс животного
  - статическое отношение **is-a** (Dog is an animal)
- Композиция
  - "часть-целое"
  - отношение **has-a**

```
class Dog extends Animal  { }

class Car {
  Engine engine;
  Wheel[] wheels;
  Door[] doors;
}
```

Java

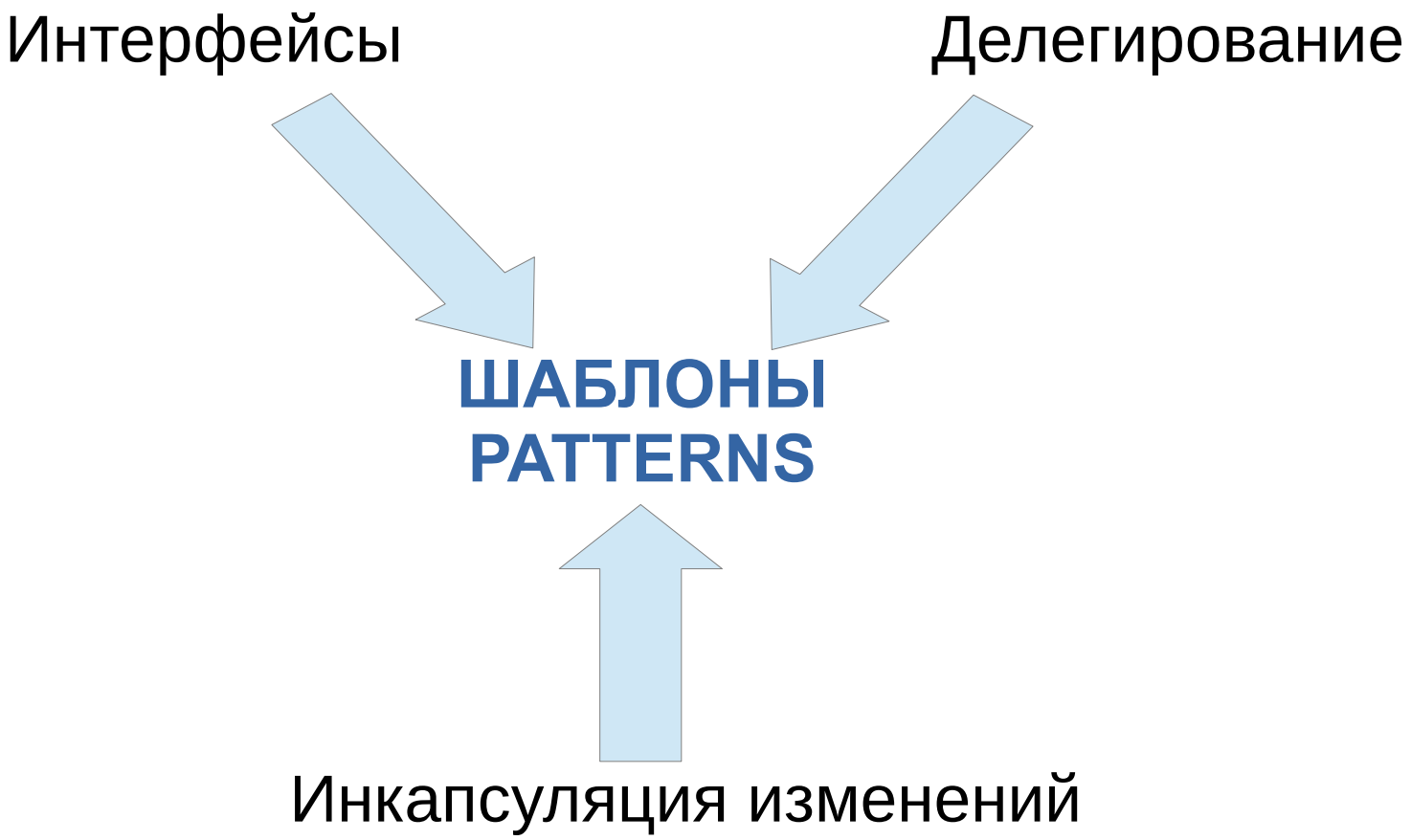
Инкапсуляция изменений

ИТМО

- Отделить изменяющееся от постоянного
- Инкапсулировать изменяющееся

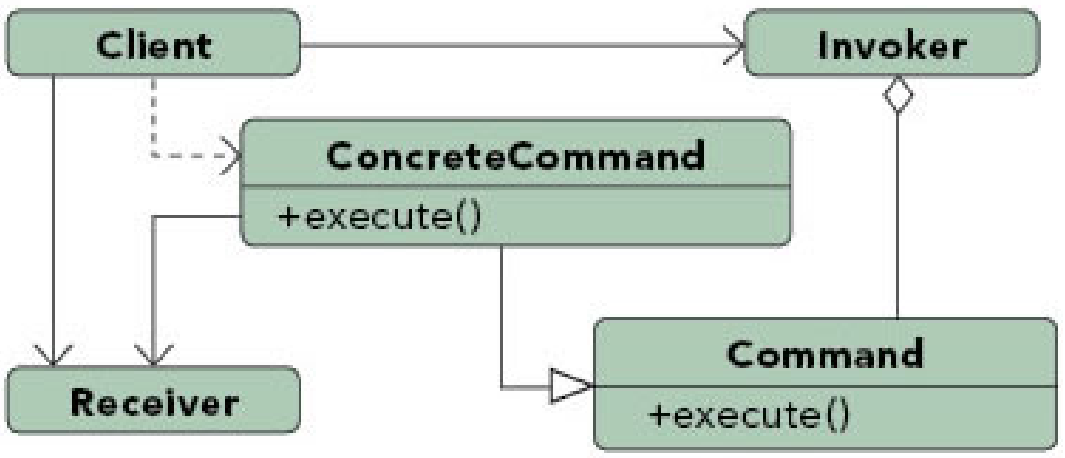
Java

Как справиться с изменениями?



Шаблон Команда (Command)

- Управление командами
- Разделение вызова и исполнения команд
- Можно организовать очередь команд и макрокоманды



Лаба 5

ИТМО

- Внимательно прочитать задание и подумать
  - Посмотреть любой вариант лаб 6, 7 и 8
- Выделить классы (не забывать SOLID)
  - Подумать над общей структурой
  - паттерн Command
- Нарисовать диаграмму
  - Объяснить уточке, как это должно работать
- Написать маленький работающий код
  - Дописывать до победы

Лаба 5 - плохой вариант

```
Scanner sc = new Scanner(System.in);
String line = sc.next();
String[] tokens = line.split(" ");
if (tokens[0].equals("help") {
    doHelp();
}
if (tokens[0].equals("add") {
    doAdd();
}
...

public void doHelp() {
    System.out.println("help - помощь");
    System.out.println("add - добавить элемент");
}
```

Лаба 5 - более правильный вариант

```
public interface Command {            // Abstract Command
    void execute();
}
public class CollectionManager {   // Receiver (исполнитель)
    List<Pokemon> pokemonList = new ArrayList<>();
    public add(Pokemon p) {
        pokemonList.add(p);
    }
}
public class AddCommand implements Command {  // Add Command
    CollectionManager cm;
    public void execute() {
        cm.add(pokemon);
    }
}
```

Лаба 5 - более правильный вариант

```
public class Invoker {
    ...
    Map<String, Command> commands = new HashMap<>();
    commands.put("help", new HelpCommand());
    commands.put("add", new AddCommand());
    ...
    Scanner sc = new Scanner(System.in);
    while (sc.hasNext()) {
        String line = sc.next();
        String[] tokens = line.split(" ");
        Command command = commands.get(tokens[0]);
        command.execute();
    }
```

Лаба 5 - более правильный вариант

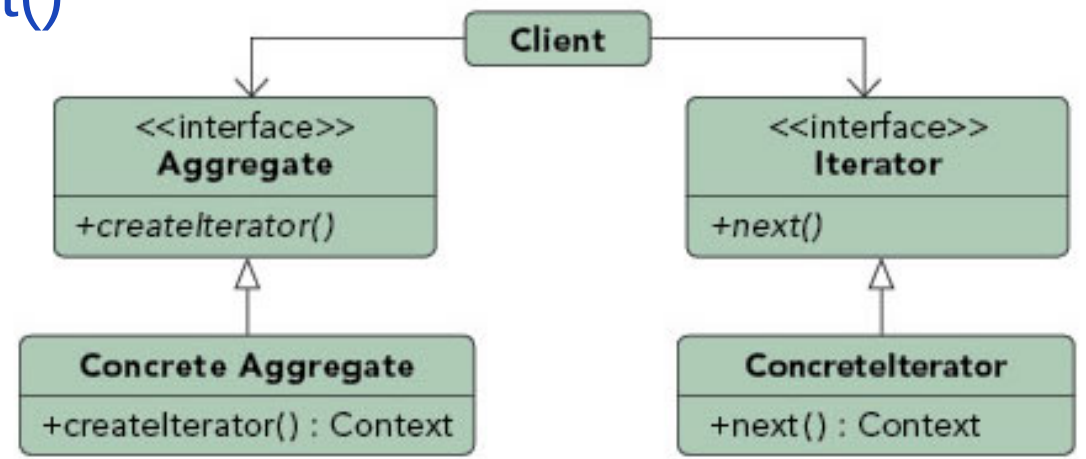
```
public interface Command {              // Abstract Command
    void execute();
    String descr();
}

public class HelpCommand implements Command {
    public String descr() { return "help - помощь"; }
    public void execute() {
        for (Command c : commands.values()) {
            System.out.println(c.descr());
        }
}
```

Виды шаблонов

- Порождающие
  - Singleton, Factory Method, Builder, ...
- Структурные
  - Adapter, Decorator, Proxy
- Поведенческие
  - Command, Iterator, Observer, Strategy

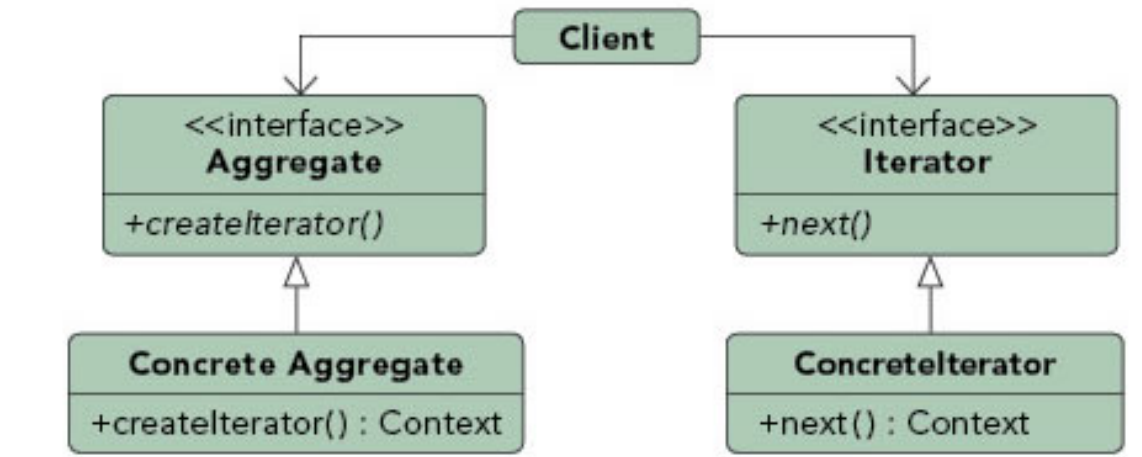
Шаблон Iterator

- Последовательный доступ к элементам
- Collection.iterator()
  - Iterator.hasNext(), .next()
- Scanner.hasNext(), .next()



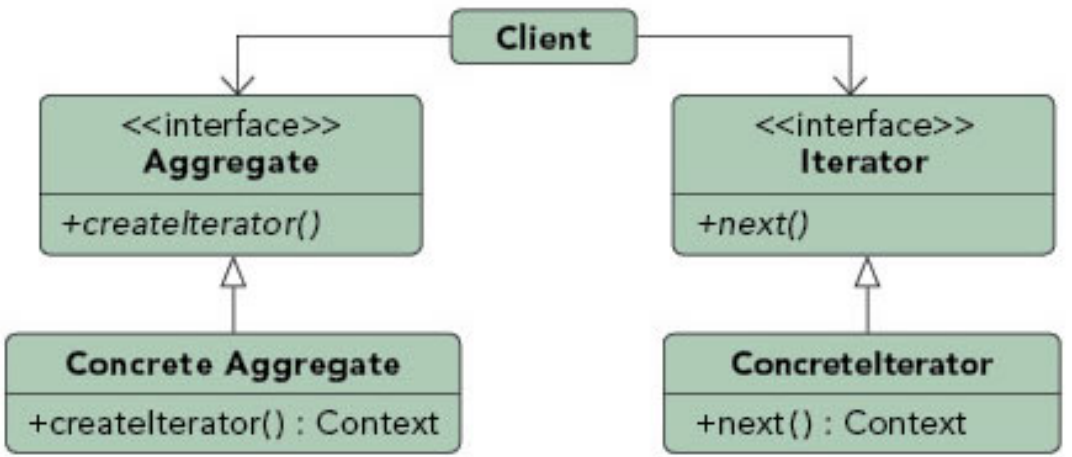
Шаблон Iterator

- Последовательный доступ к элементам коллекции
- Экскурсия: итератор - список достопримечательностей
  - Реальный гид
  - Аудиогид
  - Путеводитель



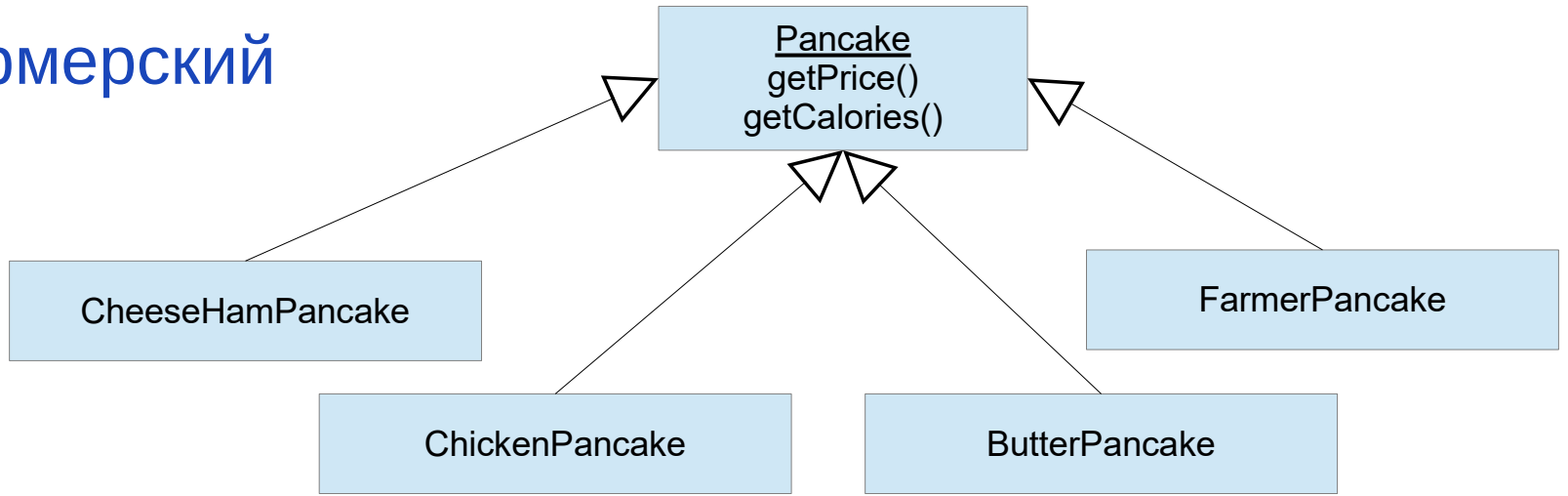
Iterator

- Универсальный доступ ко всем элементам коллекций
- Гибкая реализация обхода коллекций
- Более сложный вариант, чем простой цикл



Шаблон Decorator - Теремок

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



Шаблон Decorator - Теремок

ИТМО

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

- Добавки
  - Огурцы
  - Лук-фри
  - Картофельное пюре
  - Сметана

- ChickenPancakeWithPicklesAndMashedPotatoes
- FarmerPancakeWithSourCreamAndOnionAndPickles

Шаблон Decorator - Теремок

ИТМО

- 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

ИТМО

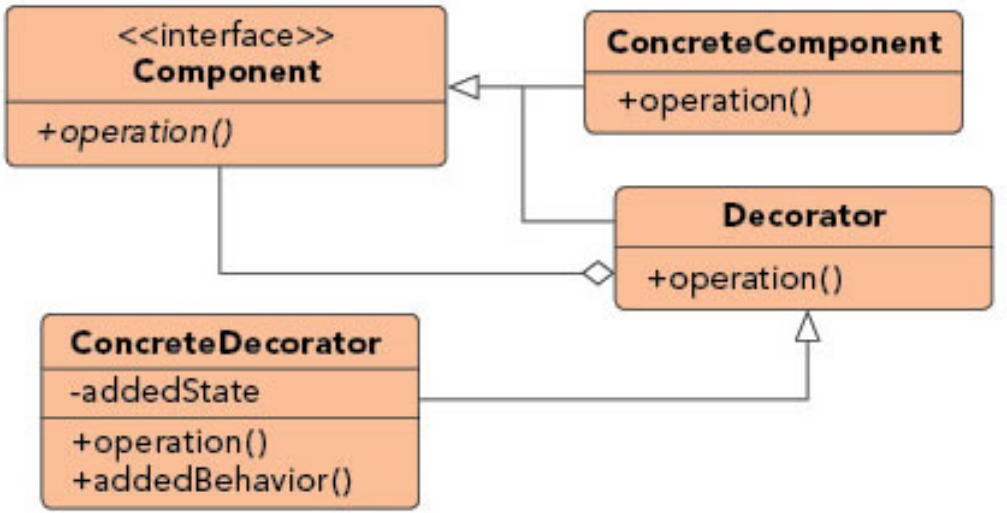
- Блин с ветчиной и сыром
- Блин с куриной грудкой
- Блин с маслом
- Блин Фермерский

- Огурцы
- Лук-фри
- Картофельное пюре
- Сметана

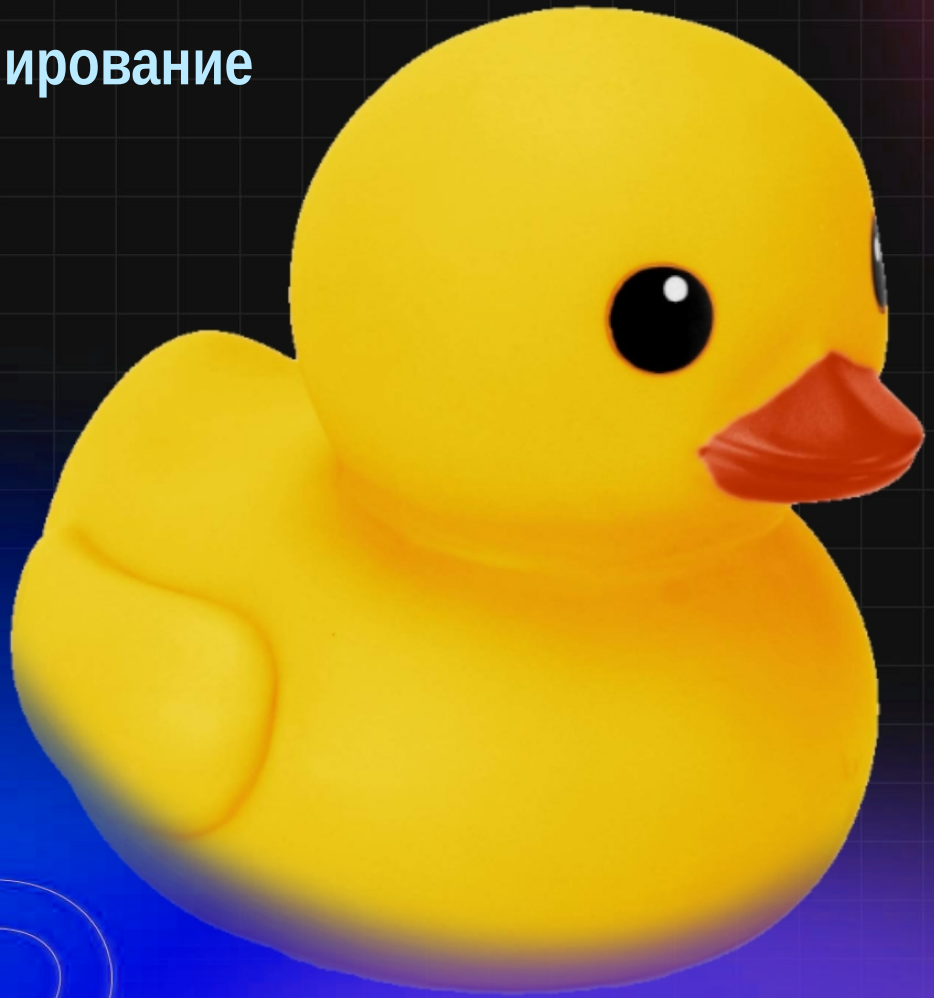
```
Pancake p =
    new MashedPotatoPancake(
        new PicklePancake(
            new CheeseHamPancake()));

p.getPrice();
```

Шаблон Decorator

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



Программирование
2 семестр
2024



ИТМО

Ввод-вывод (продолжение)

Потоки-фильтры - декораторы

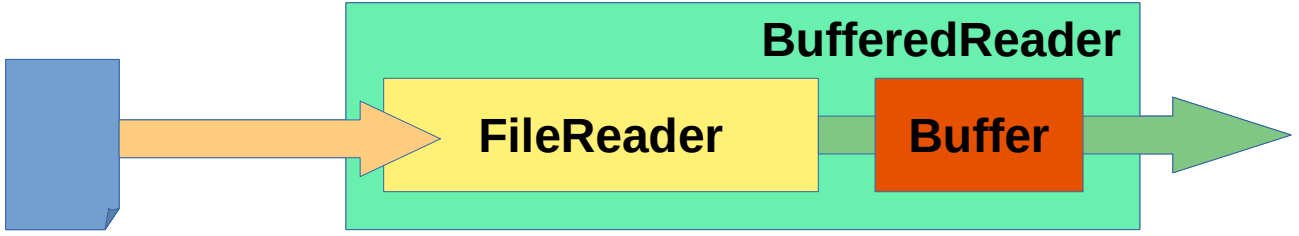
ИТМО

- FilterInputStream(InputStream)
- FilterOutpurStream(OutpurStream)
- FilterReader(Reader)
- FilterWriter(Writer)
  - исходный поток = аргумент конструктора
  - поток-фильтр = декоратор

Java

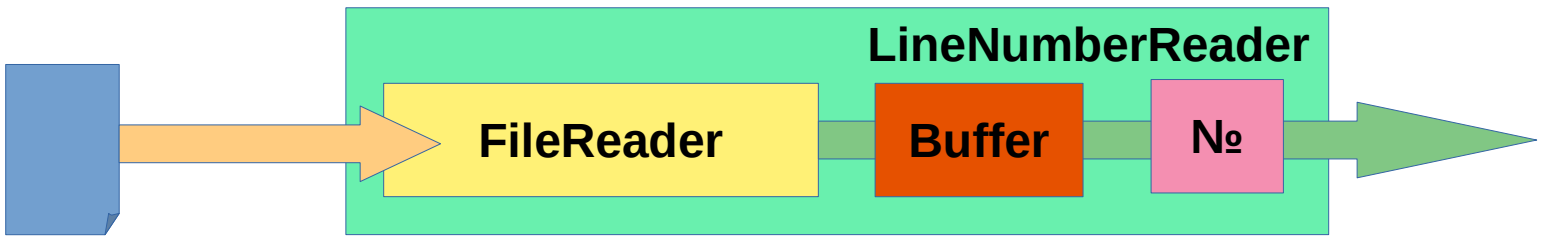
Потоки-фильтры - Buffered

- BufferedInputStream
- BufferedReader
- BufferedOutpurStream
- BufferedWriter
- Буфер для повышения производительности
- Возможность построчной работы



Потоки-фильтры - LineNumber

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



Чтение и запись с помощью потоков

ИТМО

```
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);
}
```

Java

☑ Потоки-фильтры - байт ↔ символ

ИТМО

- 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*)

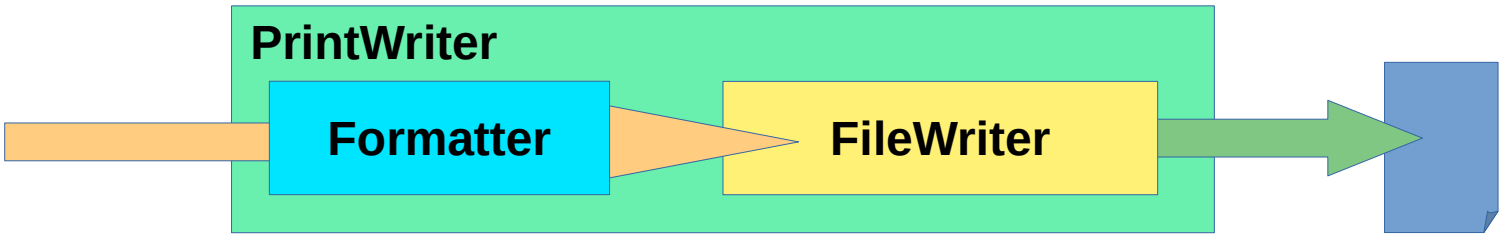
Java

PrintStream, PrintWriter

ИТМО

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

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

π = +3,1415927
```



Java

PrintStream, PrintWriter

ИТМО

- %[индекс$][флаги][размер][.точность]формат
- %b - boolean
- %h - hashcode
- %t = time/date
- %% - %
- %n - newline
- %d - decimal
- %o - octal
- %h - hex
- %s - string
- %c - char
- %f - float
- %e - exponent
- %g - %e / %f
- %a - hex float

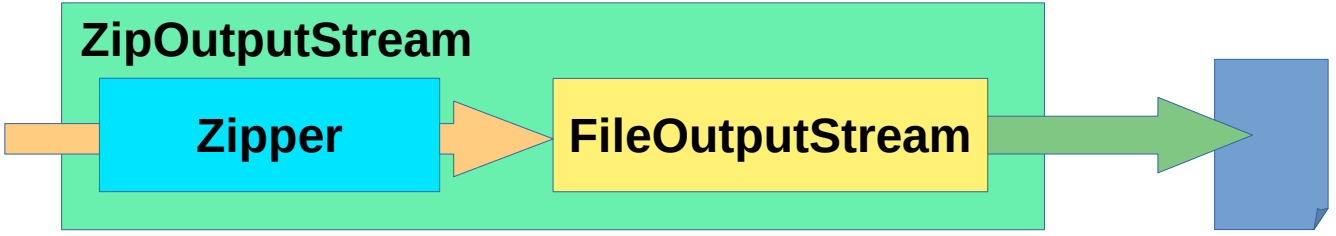
- индекс – номер аргумента
- флаги – зависят от формата
- размер – количество символов
- точность – после запятой

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

π = +3,1415927
```

Потоки-фильтры - Inflater/Deflater

ИТМО

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

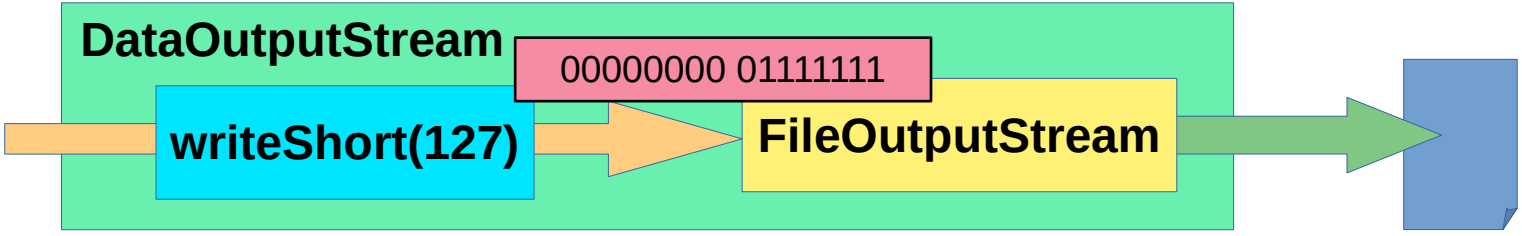


Java

DataInputStream, DataOutputStream

ИТМО

- 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()



Java

Сериализация объектов

ИТМО

- 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

Java

ObjectOutputStream, ObjectInputStream

ИТМО

- 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

Проблемы стандартной сериализации

ИТМО

- 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

Java

Замена стандартной сериализации

ИТМО

- 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

Замена стандартной сериализации

ИТМО

- 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

Scanner

ИТМО

- 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)

Java

Консольный ввод-вывод

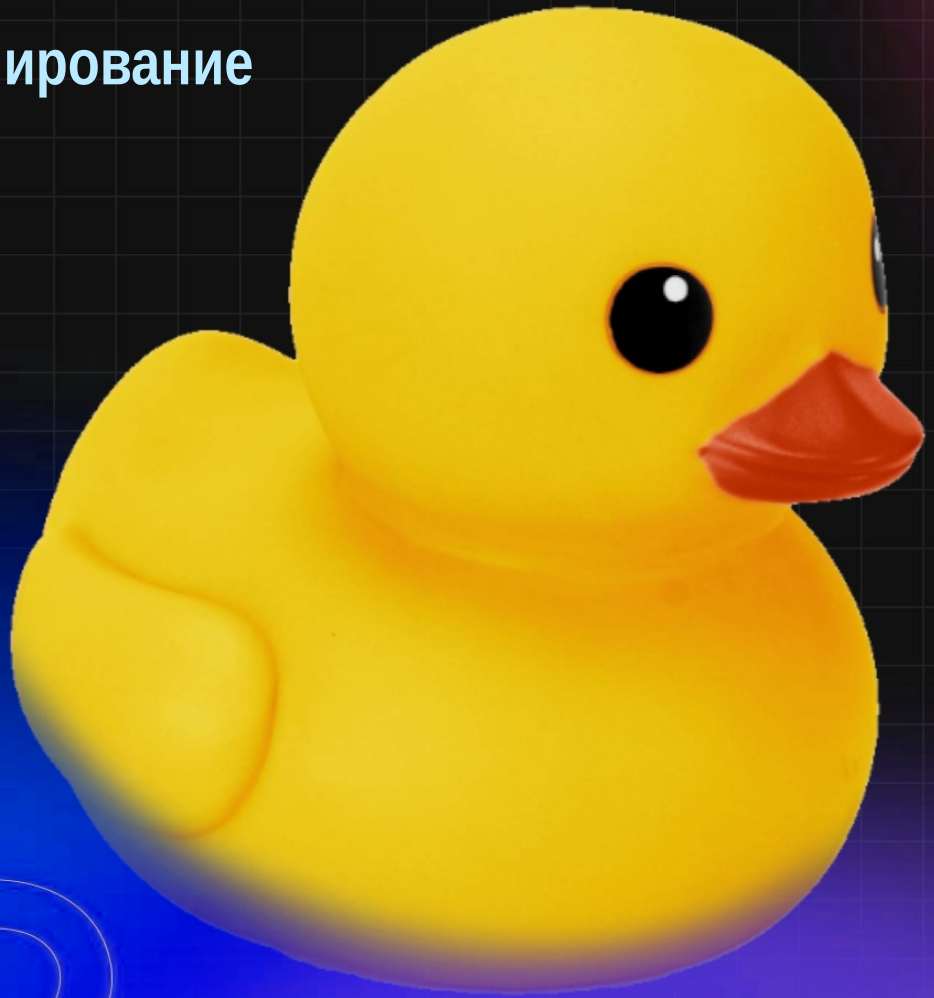
ИТМО

- 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()

Java

Программирование
2 семестр
2024

ИТМО



Вспомогательные классы

Класс java.util.Random

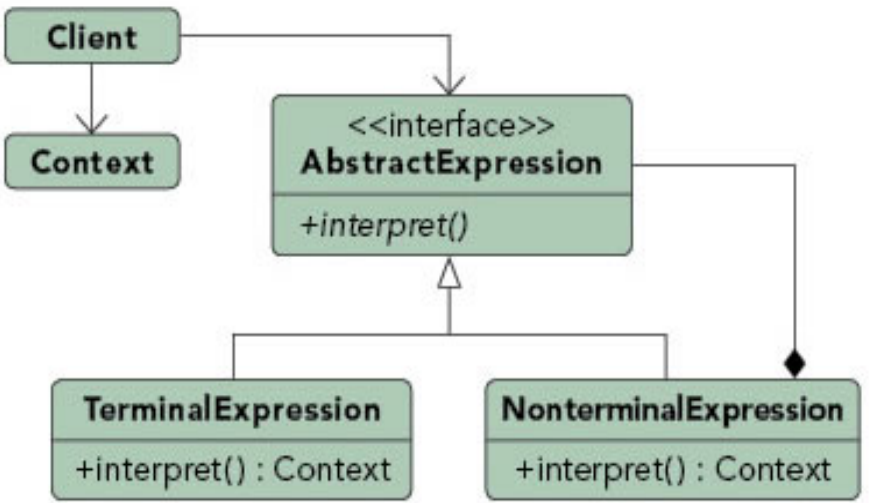
ИТМО

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

Java

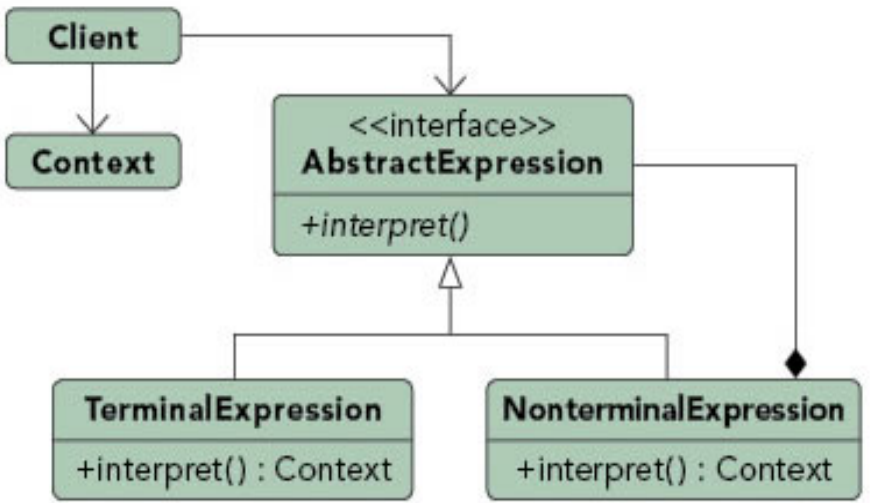
Interpreter

- Позволяет управлять поведением с помощью простого языка
  - Регулярные выражения
  - Форматирование строк



Interpreter

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



Регулярные выражения

ИТМО

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

```
String regex = "a*b";
Pattern p = Pattern.compile(regex);
Matcher m = p.matcher("aaabbb");
boolean b = m.matches();

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

Регулярные выражения

ИТМО

```
x - x
\\ - \
\0nnn - 8-ричный код (байт)
\xhh  - 16-ричный код (байт)
\uhhhh - 16-ричный код (Unicode)
\t - tab
\n - newline
\r - carriage-return
```

Регулярные выражения

ИТМО

подстрока символов

```
in
```

начало и конец строки

```
^lo
```

```
ua$
```

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

Java

Регулярные выражения

ИТМО

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
```

Java

Регулярные выражения

ИТМО

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
```

Java

Регулярные выражения

ИТМО

любой символ

```
...c
```

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

Java

Регулярные выражения

ИТМО

альтернатива

```
s(i|ec)t
```

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

Java

Регулярные выражения

ИТМО

квантификаторы

```
\s.{6}\s
e.{0,2}i
s\S{0,1}m
```

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

Java

Регулярные выражения

ИТМО

квантификаторы

s\S?m	{0,1}
b.+d	{1,}
et[^o]*,	{0,}

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

Java

Регулярные выражения

ИТМО

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
```

Java

Регулярные выражения

ИТМО

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
```

Регулярные выражения

ИТМО

группы

```
(\w)..\1(.).{1,6}\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
```

Java

Запись и чтение CSV, JSON, XML

ИТМО

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

Java

Запись и чтение CSV, JSON, XML

ИТМО

- 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

Запись и чтение CSV, JSON, XML

ИТМО

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

Запись и чтение CSV, JSON, XML

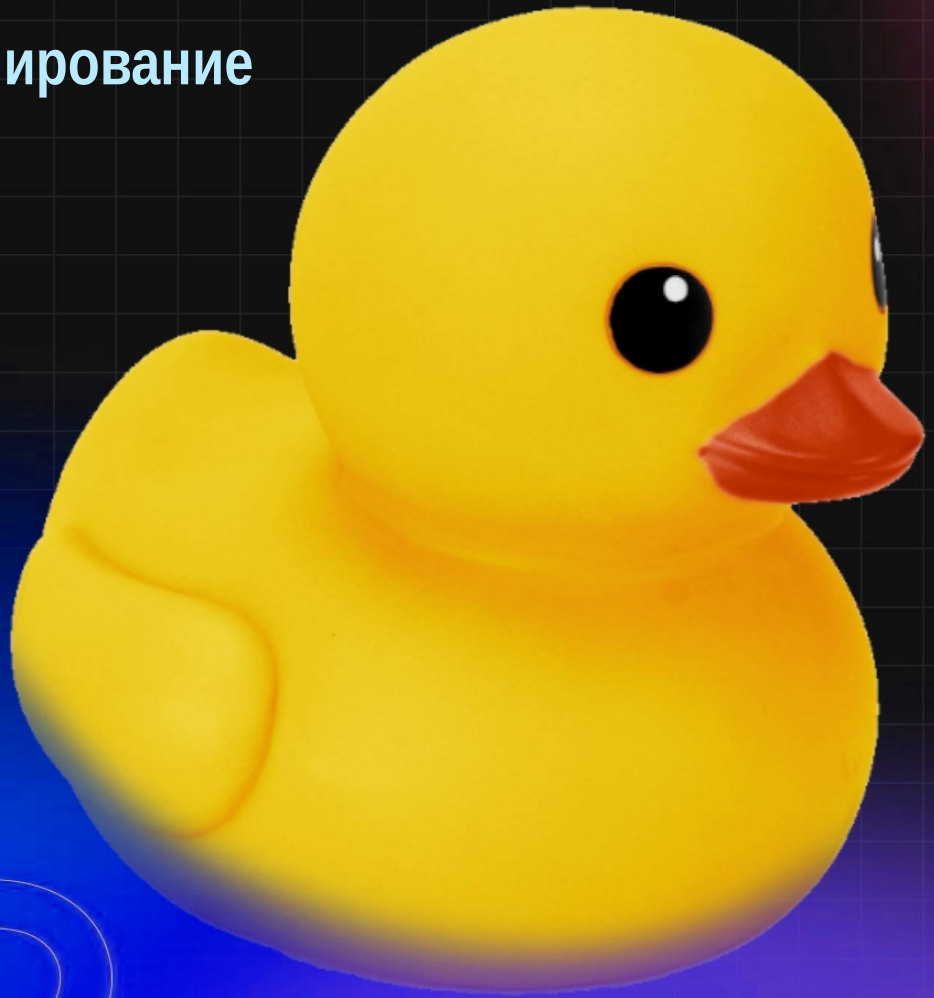
ИТМО

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

Javadoc

```
/**
 * 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
2024



Файлы (IO / NIO.2)

Файлы и файловые системы

ИТМО

- Файловая система
  - Каталоги
  - Файлы
    - Жесткие ссылки
    - Символические ссылки
- Путь к файлу
  - абсолютный (от корня)
  - относительный

- Windows (NTFS)
  - раздельные FS
  - разделитель - \
  - C:\Users\student\file.txt
- Linux / UNIX / MacOS
  - виртуальная FS
  - разделитель - /
  - /home/student/file.txt

Java

Файлы

ИТМО

- 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 Path.toFile()`
  - `Path File.toPath()`

Java

Файловые системы

ИТМО

- abstract class java.nio.file.FileSystem — файловая система
  - Методы:

```
Iterable<Path> getRootDirectories()
Path getPath(String first, String... more)
String getSeparator()
```

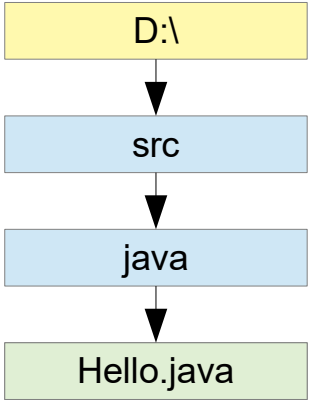
- класс-утилита java.nio.file.FileSystems

```
FileSystem getDefault()
```

Java

Пути к файлам

- 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");
```

Действия с файлами: проверки

ИТМО

- 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)
```

Java

Работа с файлами: create/delete

ИТМО

```
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

ИТМО

```
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

чтение

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

запись

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