Язык Java

Глава IV. Шаблоны, коллекции, потоки

Виталий Витальевич Перевощиков

Осенний семестр 2022

Содержание

1. Шаблоны (Generics)

2. Функциональные интерфейсы и лямбда-выражения

3. Коллекции

4. Потоки (Streams)

Шаблоны (Generics)

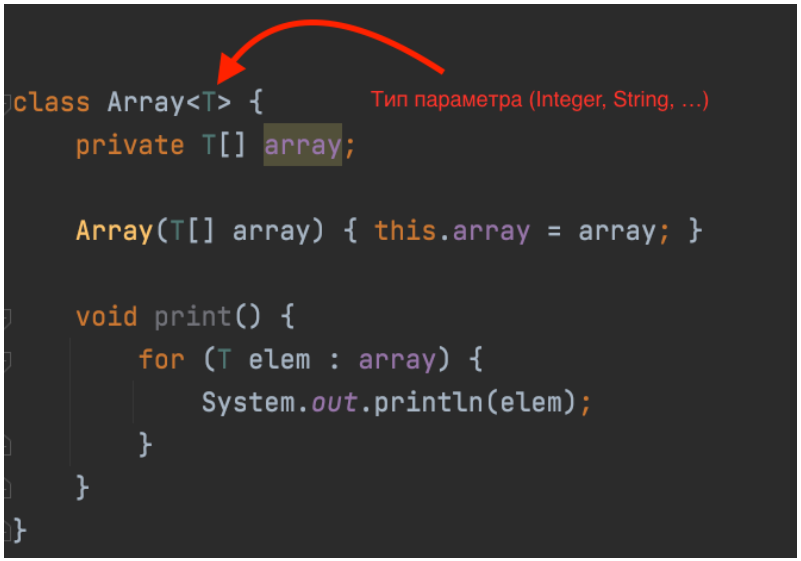
- Шаблоны - это параметризованные типы
- Мотивация: сократить повторение кода:

```
class IntegerArray {
    private Integer[] array;

    IntegerArray(Integer[] array) { this.array = array; }

    void print() {
        for (Integer elem : array) {
            System.out.println(elem);
        }
    }
}

class StringArray {
    private String[] array;

    StringArray(String[] array) { this.array = array; }

    void print() {
        for (String elem : array) {
            System.out.println(elem);
        }
    }
}
```

- Можно ли ввести переменную для типа массива? Да!

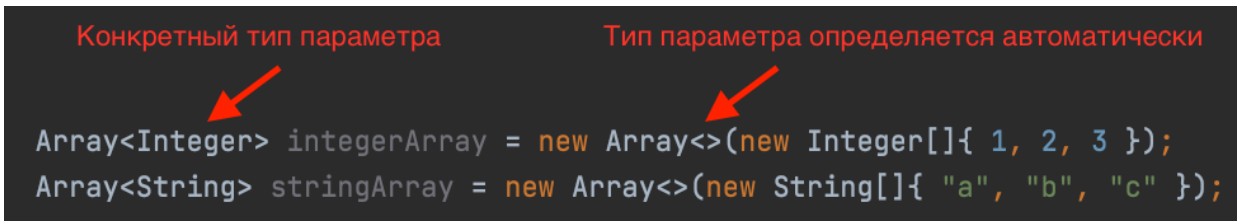
Шаблонные классы

- Определение шаблонного класса:




```
class Array<T> {
    private T[] array;

    Array(T[] array) { this.array = array; }

    void print() {
        for (T elem : array) {
            System.out.println(elem);
        }
    }
}
```

- Использование шаблонного метода:




```
Array<Integer> integerArray = new Array<>(new Integer[]{ 1, 2, 3 });
Array<String> stringArray = new Array<>(new String[]{ "a", "b", "c" });
```

- **Внимание:**
  - Тип параметра по умолчанию: `Object`

```
// то же самое, что Array<Object>
Array objectArray = new Array(new Object[]{ 1, "a" });
```

  - Тип параметра не может быть примитивным (`int`, `double`, ...)

Шаблонные классы

- Шаблонные классы с несколькими параметрами:

```
class Pair<T, U> {
    private T firstElement;
    private U secondElement;

    Pair(T firstElement, U secondElement) {
        this.firstElement = firstElement;
        this.secondElement = secondElement;
    }
}
```

```
Pair<String, Double> pair = new Pair<>("key", 1.2);

// то же самое, что Pair<Object, Object>
Pair objectPair = new Pair(new Object(), new Object());
```

Шаблонные методы

- **Случай 1:** Тип параметра метода совпадает с типом параметра класса:

```
class Array<T> {
    private T[] array;

    Array(T[] array) { this.array = array; }

    public T getFirstElement() {
        return array[0];
    }
}
```

```
Array<Integer> array = new Array<>(new Integer[]{1, 2, 3});
int firstElement = array.getFirstElement(); // 1
```

- **Случай 2:** Тип параметра метода отличается от типа параметра класса:

```
class IntegerConverter {
    // явное приведение переменной типа T к типу int
    public static <T> int toInteger(T value) {
        return (int)value;
    }
}
```

```
Long value = 123L;
int result = IntegerConverter.toInteger(value); // Runtime error!!!
```

Функциональные интерфейсы

- Мотивация: использование функции $f : T \to R$ как переменной
- **Функциональный интерфейс** - интерфейс `Function<T,R>` из пакета `java.util.function`
- Примеры:

```
class FunctionExamples {
    static Double squareRootStatic(Integer number) { return Math.sqrt(number); }

    Double squareRoot(Integer number) { return Math.sqrt(number); }

    void showExamples() {
        // определение функциональной переменной для функции func: Integer -> Double
        Function<Integer, Double> func;

        // присваивание значения функциональной переменной
        func = FunctionExamples::squareRootStatic; // ссылка на статический метод
        func = this::squareRoot;                   // ссылка на метод объекта
        func = x -> Math.sqrt(x);                  // лямбда-выражение

        // вычисление значения функции
        Integer argument = 4;
        Double value = func.apply(argument); // 2.0
    }
}
```

Лямбда-выражения

```
// функция одной переменной f: Integer -> String
Function<Integer, String> f;
f = a -> a.toString();
f = (Integer a) -> a.toString();
f = (a) -> { return a.toString(); };

// функция двух переменных f: Integer x Integer -> Boolean
BiFunction<Integer, Integer, Boolean> g;
g = (x, y) -> x.equals(y);
g = (Integer x, Integer y) -> { return x.equals(y); };

// Каррирование ("Currying") f: Integer -> (Integer -> Boolean)
Function<Integer, Function<Integer, Boolean>> h;
h = x -> y -> x.equals(y);
h = (Integer x) -> (Integer y) -> { return x.equals(y); };

// Вызов функций g и h
Boolean gResult = g.apply(1, 2); // false
Boolean hResult = h.apply(1).apply(2); // false
```

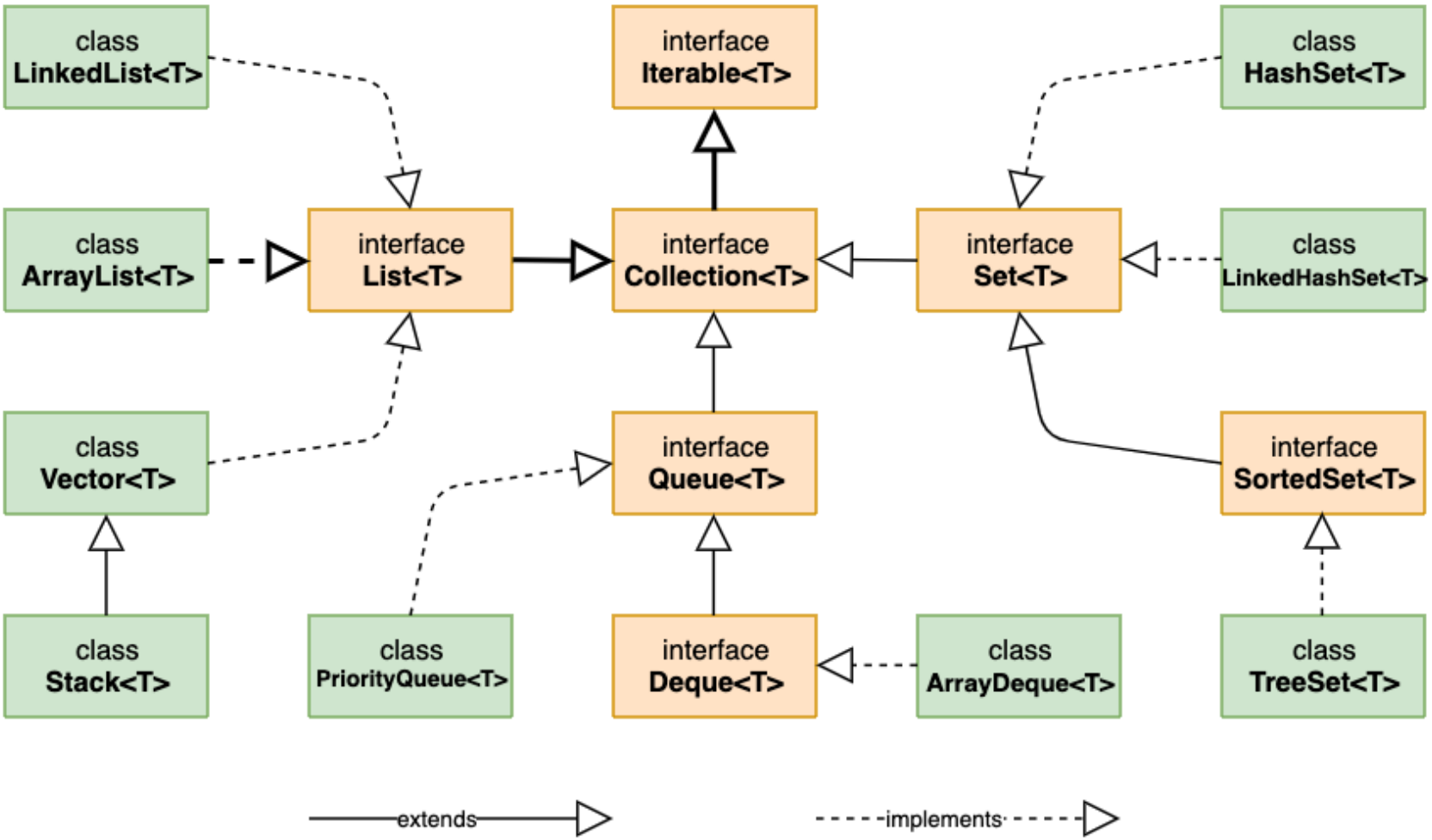
Интерфейсы Supplier<T>, Consumer<T>, Runnable

```java
// функция не имеет аргументов, но возвращает значение
Supplier<Integer> func1 = () -> 1;
Integer value = func1.get(); // 1

// функция имеет аргумент, но не возвращает значение
Consumer<Integer> func2 = x -> System.out.println(x);
func2.accept(2); // вывод 2 в консоль

// функция не имеет аргументов и не возвращает значение
Runnable func3 = () -> System.out.println(1);
func3.run(); // вывод 1 в консоль
```

Индивидуальные функциональные интерфейсы

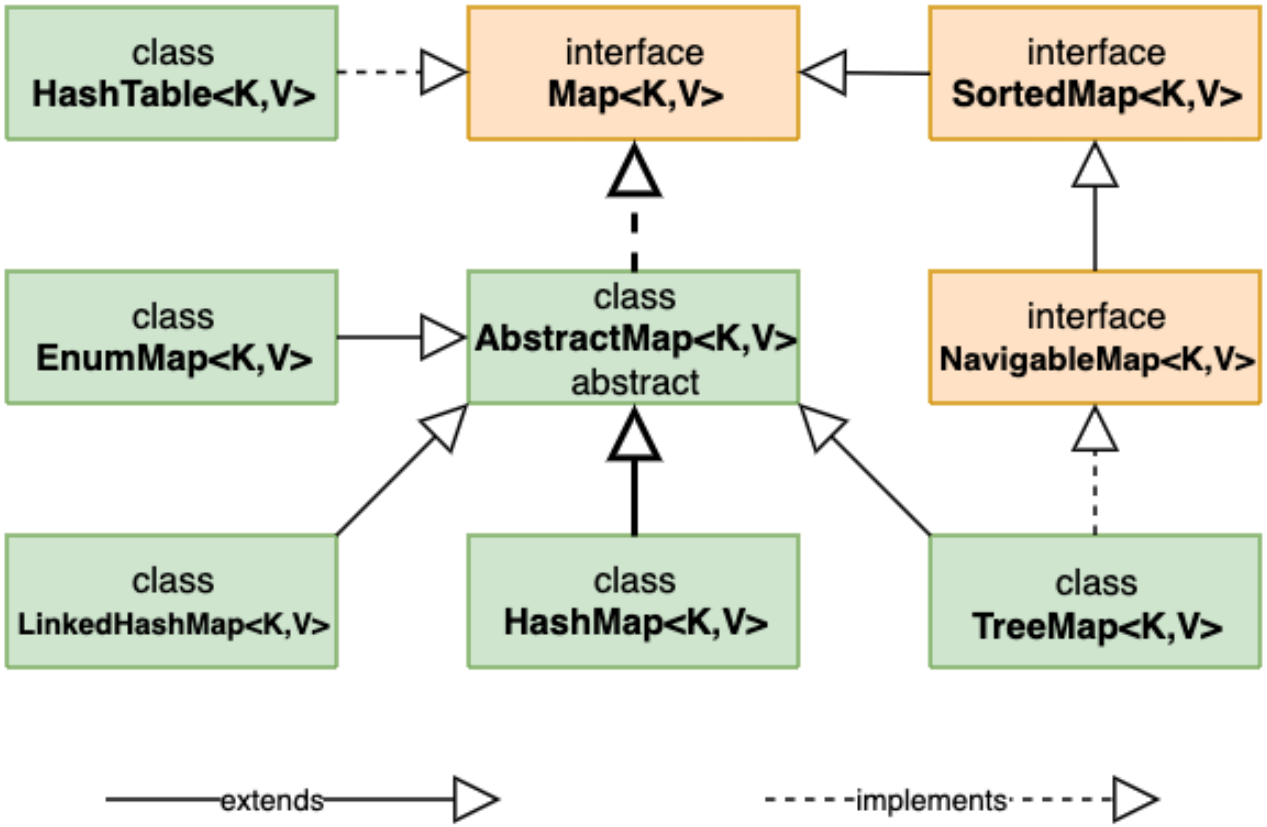
```java
// Функциональный интерфейс для функции двух переменных
@FunctionalInterface
interface BinaryOperation {
    Integer func(Integer x, Integer y);
}

class FunctionExamples {

    void showExamples() {
        BinaryOperation sum = (x, y) -> x + y;
        Integer result = sum.func(1, 2); // 3
    }
}
```

Коллекции java.util



В этом курсе мы будем работать с интерфейсом `List<T>` и его реализацией `ArrayList<T>`

Maps (словари) `java.util`



В этом курсе мы будем работать с интерфейсом `Map<K,V>` и его реализацией `HashMap<K,V>`

Интерфейс Collection<T>

```
interface Collection<T> {

    // "мутации"
    boolean add(T t);           // добавить элемент в коллекцию
    boolean remove(Object o);   // удалить элемент из коллекции
    void clear();               // очистить коллекцию

    // "запросы"
    boolean contains(Object o); // проверка наличия элемента в коллекции
    int size();                 // размер коллекции
    boolean isEmpty();          // проверка пустоты

    Iterator<T> iterator();     // порядок обхода элементов коллекции

    Object[] toArray();         // преобразование коллекции в массив

    // ...
}
```

Интерфейс List<T>

- Линейный список
- Каждый элемент списка имеет индекс (как массив)

```
interface List<T> extends Collection<T> {

    // "мутации"
    T set(int index, T element);    // изменить элемент с индексом index
    void add(int index, T element); // добавить новый элемент на позиции index
    T remove(int index);            // удалить элемент с индексом index

    // "запросы"
    T get(int index);               // значение элемента с индексом index
    int indexOf(Object o);          // индекс первого появления элемента o в списке
    int lastIndexOf(Object o);      // индекс последнего появления элемента o в списке

    List<T> subList(int fromIndex, int toIndex); // подсписок с индексами [fromIndex, ..., toIndex - 1]

    // ...
}
```

Класс `ArrayList<T>`

- Реализация списка с помощью массива (используется буфер типа `Object[]`)
- Динамическое увеличение размера буфера:
  - При переполнении буфера создается новый буфер вдвое большего размера
  - Происходит копирование элементов из старого буфера в новый
- Добавление элемента в список - “дорогая” операция?
  - Допустим, что начальная длина буфера равна 1
  - При добавлении $2^k + 1$ элементов в пустой список буфер будет удваиваться k раз (при добавлении элементов с индексами 1, 2, 2^2, 2^3 ..., 2^k)
  - При i-том удвоении буфера ($1 \leq i \leq k$) будет копироваться 2^{i-1} элементов
  - Общее число операций копирования: $\sum_{i=1}^{k} 2^{i-1} = \sum_{i=0}^{k-1} 2^i = 2^k - 1$
  - Среднее количество операций копирования на одно добавление: $\frac{2^k-1}{2^k+1} = 1 - \frac{2}{2^k+1} = O(1)$

Сложность методов ArrayList<T>

Метод	Сложность
`add(T t)`	$O(1)$ (в среднем)
`add(int index, T t)`	$O(n)$
`get(int index)`	$O(1)$
`remove(int index)`	$O(n)$
`remove(Object o)`	$O(n)$
`indexOf(Object o)`	$O(n)$
`contains(Object o)`	$O(n)$

n - длина списка

Примеры с List<T>

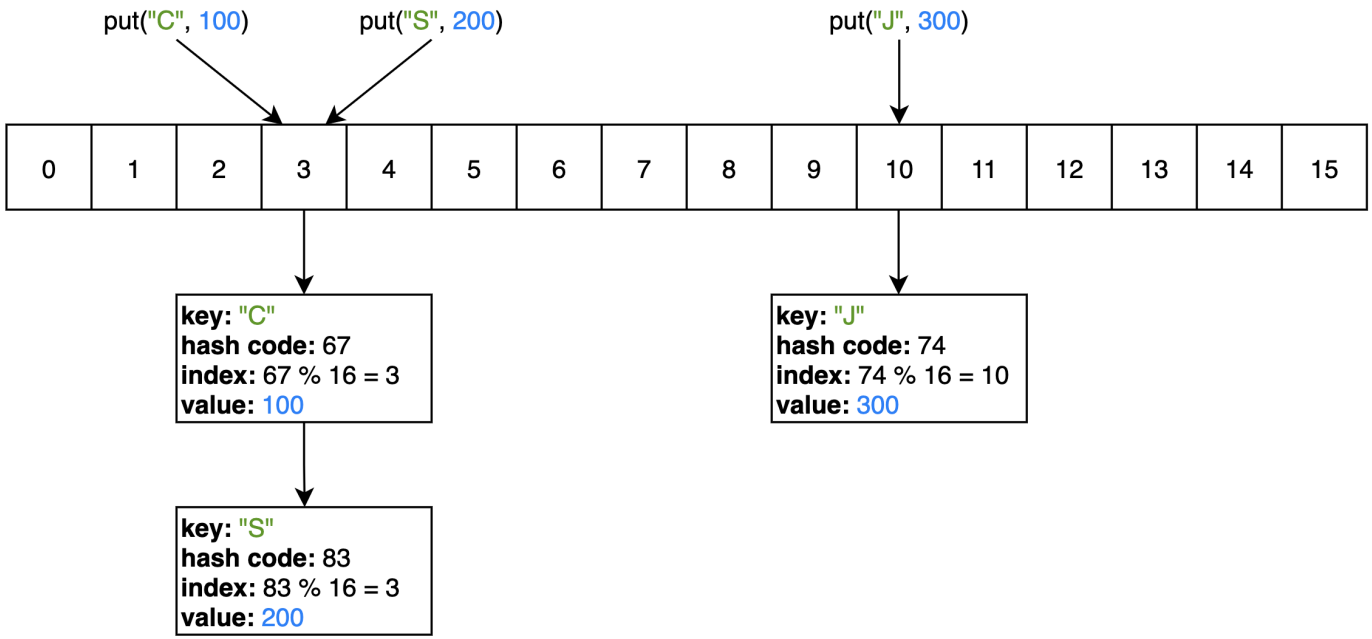
```java
// инициализация списка
List<String> list = new ArrayList<>(); // создание пустого списка
list = new ArrayList<>(20); // создание списка и выделение памяти для 20 элементов

// добавление элементов в конец списка
list.add("A");      // list = ["A"]
list.add("B");     // list = ["A", "B"]
list.add("A");     // list = ["A", "B", "A"]
list.add(2, "C"); // list = ["A", "B", "C", "A"]

// запросы
String firstItem = list.get(0);           // "A"
int firstIndexOfA = list.indexOf("A");    // 0
int lastIndexIfA = list.lastIndexOf("A"); // 3
int size = list.size();                   // 4

// подсписок
List<String> subList = list.subList(1, 3); // sublist = ["B", "C"]

// удаление элементов
list.remove("B"); // list = ["A", "C", "A"]
list.remove(2); // list = ["A", "C"]

// обход списка
for (String item : list) {
    System.out.print(item + " "); // A C
}

// очистка списка
list.clear(); // list = []
boolean isEmpty = list.isEmpty(); // true
```

Интерфейс Map<K,V>

```
interface Map<K,V> {
    // "мутации"
    V put(K key, V value);
    V remove(Object key);
    public void clear();

    // запросы
    V get(Object key); // получить значение по ключу
    boolean containsKey(Object key);
    boolean containsValue(Object value);

    // размер
    int size();
    boolean isEmpty();

    // коллекции
    Set<K> keySet(); // множество ключей
    Collection<V> values(); // коллекция значений
    Set<java.util.Map.Entry<K, V>> entrySet(); // множество пар "ключ-значение"
}
```

Класс HashMap<K,V>

- Реализация в виде массива связных списков
  - для ключа `key` вычисляется хеш: `int hash = key.hashCode()`
  - индекс массива вычисляется как:
    `int index = hash % arrayLength`
- Пример: `HashMap<String, Integer>`



- Сложность операций:
  - $O(n)$ в худшем случае
  - $O(1)$ в среднем

Примеры с Map<K,V>

```java
// инициализация
Map<String, Integer> map = new HashMap<>();

// добавление пар "ключ-значение"
map.put("C", 100);
map.put("B", 200);
map.put("A", 300);
map.put("C", 400); // изменение значения для "C"

int size = map.size(); // 3
int cValue = map.get("C"); // 400
boolean hasKey = map.containsKey("C"); // true
boolean hasValue = map.containsValue(400); // true

// метод forEach с лямбда-выражением
map.forEach((key, value) -> {
    System.out.print(key + ":" + value + " ");
}); // A:300 B:200 C:400 (не в порядке добавления)
```

Java 8 Stream API

- Stream API - библиотека для обработки последовательностей элементов (потоков)
- Поток не является структурой данных, т.е., не хранит никакие данные
- Поток имеет свой `источник данных` (например, `List` или `Set`)
- Поток не модифицирует свой источник данных, а генерирует новые данные
- Операции над потоками:
  - **Промежуточные:** преобразуют один поток в другой
  - **Конечные:** преобразуют поток в данные (например, `List` или `Set`)
- Интерфейс для потоков: `Stream<T>` из пакета `java.util.stream`

Создание потока

```java
// перечисление элементов
Stream<Integer> stream = Stream.of(1, 2, 3, 4);

// источник данных: массив
Integer[] array = {1, 2, 3, 4};
stream = Stream.of(array);

// источник данных: коллекция
Collection<Integer> collection = List.of(1, 2, 3, 4);
stream = collection.stream();

// генерация "бесконечной" случайной последовательности чисел
Random random = new Random(); // генератор случайных чисел
stream = Stream.generate(() -> random.nextInt());
```

Промежуточные операции над потоками

- `filter`

```
Stream<Integer> stream = Stream.of(1, 2, 3, 4, 5, 6);
// поток четных чисел: 2, 4, 6
Stream<Integer> evenNumbersStream = stream.filter(number -> number % 2 == 0);
```

- `map`

```
Stream<Integer> stream = Stream.of(1, 2, 3, 4, 5, 6);
// увеличение чисел на 1
Stream<Integer> incrementStream = stream.map(number -> number + 1); // поток 2, 3, 4, 5, 6, 7
```

- `flatMap`

```
Integer[] firstArray = {1, 2, 3};
Integer[] secondArray = {4, 5, 6};
Integer[] thirdArray = {7, 8, 9};
Stream<Integer[]> stream = Stream.of(firstArray, secondArray, thirdArray); // поток [1, 2, 3], [4, 5, 6], [7, 8, 9]
// объединение массивов в один
Stream<Integer> mergeStream = stream.flatMap(array -> Stream.of(array)); // поток 1, 2, 3, 4, 5, 6, 7, 8, 9
```

Промежуточные операции над потоками

- skip

```
Stream<Integer> stream = Stream.of(1, 2, 3, 4, 5, 6);
// пропуск первых трех элементов потока
Stream<Integer> skipStream = stream.skip(3); // поток 4, 5, 6
```

- limit

```
Stream<Integer> stream = Stream.of(1, 2, 3, 4, 5, 6);
// ограничение потока до 3 первых элементов
Stream<Integer> limitStream = stream.limit(3); // поток 1, 2, 3
```

- distinct

```
Stream<Integer> stream = Stream.of(1, 2, 3, 4, 5, 6);
// ограничение потока до 3 первых элементов
Stream<Integer> limitStream = stream.limit(3); // поток 1, 2, 3
```

- sorted

```
Stream<Integer> stream = Stream.of(4, 1, 2, 6, 5, 3);
// сортировка элементов потока по возрастанию
Stream<Integer> sortedStream = stream.sorted(); // поток 1, 2, 3, 4, 5, 6

// сортировка элементов потока по убыванию
Stream<Integer> reverseSortedStream = stream.sorted(Comparator.reverseOrder()); // поток 6, 5, 4, 3, 2, 1
```

Сортировка по нескольким параметрам

- Сортировка потока студентов по имени, а затем по номеру семестра (если имена совпадают)

```java
class Student {
    private String name;
    private int semester;

    public Student(String name, int semester) {
        this.name = name;
        this.semester = semester;
    }

    public String getName() {
        return this.name;
    }

    public int getSemester() {
        return this.semester;
    }
}
```

```java
Stream<Student> stream = Stream.of(
        new Student("Петров", 1),
        new Student("Иванов", 1),
        new Student("Иванов", 2)
);
// сортировка сперва по имени, а потом по семестру
Stream<Student> sortedStream = stream.sorted(Comparator
        .comparing(Student::getName)
        .thenComparing(Student::getSemester)); // поток {"Иванов", 1}, {"Иванов", 2}, {"Петров", 3}
```

Конечные операции над потоками

- `collect`
  - преобразование в список

```java
Stream<Integer> stream = Stream.of(1, 2, 3, 4, 5, 6);

// преобразование потока в список
List<Integer> list = stream.collect(Collectors.toList()); // реализация неизвестна
// использование конкретной реализации списка ArrayList
ArrayList<Integer> arrayList= stream.collect(Collectors.toCollection(ArrayList::new));
```

  - преобразование в словарь

```java
class Teacher {
    private int id; // уникальный идентификатор
    private String name;

    public Teacher(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }
}
```

```java
Stream<Teacher> stream = Stream.of(
        new Teacher(1, "Иванов"),
        new Teacher(2, "Петров"),
        new Teacher(3, "Сидоров"));

// словарь с ключом id и значением name
Map<Integer, String> studentMap = stream.collect(Collectors.toMap(Teacher::getId, Teacher::getName));
// словарь 1:Иванов, 2:Петров, 3:Сидоров
```

Конечные операции над потоками

- reduce

```
Integer[] array = { 1, 2, 3, 4, 5, 6 };

// вычисление суммы с помощью цикла for-each
int sum = 0;
for (Integer item : array) {
    sum = sum + item;
}

// вычисление суммы с помощью reduce
Stream<Integer> stream = Stream.of(array);
sum = stream.reduce(0, (accumulated, current) -> accumulated + current);
sum = stream.reduce(0, Integer::sum); // более короткая запись
```

- forEach

```
Random random = new Random(); // генератор случайных чисел
Stream<Integer> stream = Stream.generate(() -> random.nextInt());

// бесконечный цикл, выводящий в консоль случайные числа
stream.forEach(number -> System.out.println(number));
```