4. Вспомогательные классы

Функциональные интерфейсы и лямбда-выражения

- λ-выражение — блок кода, представляющий из себя ссылку на анонимную функцию
- Может использоваться на месте ссылки на функциональный интерфейс
- Функциональный интерфейс — интерфейс, в котором определен один и только один абстрактный метод (не считая default и методов Object)

Пример

```
@FunctionalInterface interface Comparator<T> {
    public int compare(T obj1, T obj2);
}

public void sort(Comparator<T>) // метод сортировки

class X implements Comparator<String> {
    public int compare(String s1, String s2) {
        return s1.length() - s2.length();
    }
}
sort(new X());
```

Ссылка на метод

- Если единственное, что делает λ-выражение — вызывает уже существующий метод, его можно заменить на ссылку на метод (method reference)
- `(s) -> A.method(s) ~ A::method`
- `(s) -> a.method(s) ~ a::method`
- `(s) -> new A(s)    ~ A::new`
- `(s) -> s.method()  ~ S::method`

Ссылка на метод (пример)

```
@functionalInterface interface F { String trans(String s) }

class A {
  public static String upper(String s) { return s.toUpperCase();
}
  public String lower(String s) { return s.toLowerCase(); }
  public static void make(F t, String s) { return t.trans(s); }
  public static void main(String[] args) {
    A a = new A();
    make((s) -> upper(s), "Hello");
    make(A::upper, "Hello"); // A.upper("Hello")
    make(a::lower, "Hello"); // a.lower("Hello")
    make(String::new, "Hello"); // new String("Hello")
    make(String::toLowerCase, "Hello"); // (s) -> s.toLowerCase()
  }
}
```

Функциональные интерфейсы

- java.lang.Runnable
- java.util.Comparator
- java.util.function.* - набор функциональных интерфейсов общего назначения для разных случаев

Пакет java.util.function

- `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) }`

Пакет java.util.function

- int, long, double
  - IntSupplier { int getAsInt() }
  - IntConsumer { void accept(int value) }
  - IntPredicate { boolean test(int value) }
  - IntFunction<R> { R apply(int value) }
  - IntUnaryOperator { int applyAsInt(int value) }
  - IntBinaryOperator { int applyAsInt(int v1, int v2) }
  - ToIntFunction<T> { int applyAsInt(T t) }
  - ToIntBiFunction<T,U> { int applyAsInt(T t, U u) }

Пакет java.util.stream

- Конвейерная обработка данных
- Поток — последовательность элементов
- Поток может быть последовательным или параллельным
- Конвейер — последовательность операций
- Отличия от коллекций
  - Элементы не хранятся
  - Неявная итерация
  - Функциональный стиль — операции не меняют источник
  - Большинство операций работают с λ-выраженями
  - Ленивое выполнение
  - Возможность неограниченного числа элементов

Операции над потоком элементов

- Конвейер состоит из источника (коллекция, массив, фабрика элементов, канал ввода-вывода, ...), нескольких (0 или больше) промежуточных операций и одной завершающей операции.
- Способы получения потока из источника:
- Collection.stream()
- Collection.parallelStream()
- Arrays.stream(Object[])
- Stream.of(Object[])
- IntStream.range(int, int)
- ...

ИТМО ВТ

Промежуточные и завершающие операции

- Промежуточные операции возвращают поток
- Промежуточные операции выполняются "лениво" — фактически выполнение операции может быть задержано.
- Промежуточные операции делятся на:
  - Не хранящие состояние (stateless) — выполняются над элементом вне зависимости от других элементов
  - Хранящие состояние (stateful) — выполнение зависит от других элементов (например, сортировка)
- Завершающие операции возвращают некий результат, либо имеют побочное действие. После завершающей операции поток прекращает существование.

Классы и интерфейсы

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

Spliterator

- Параллельный Iterator
- Spliterator trySplit()
- void forEachRemaining(Consumer action)
- boolean tryAdvance(Consumer action)

Классы и интерфейсы

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

Классы и интерфейсы (продолжение)

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

Терминальные операции

- void forEach(Consumer<T> action)
- void forEachOrdered(Consumer<T> action)
  - выполняет действие для каждого элемента потока
  - второй вариант гарантирует сохранение порядка элементов
- Optional<T> min(), Optional<T> max()
  - возвращают минимальный и максимальный элементы,
- long count(), int (long, double) sum()
  - возвращают количество и сумму элементов
- reduce(ident, BiFunction acc, BinaryOperattor comb)
  - операция редукции — аккумулятор заменяется
- collect(Supplier supp, BiFunction acc, BinaryOperattor comb)
  - операция изменчивой редукции — аккумулятор обновляется

Проверки

- 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

Пример

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

Класс java.util.Optional<T>

- Оболочка, которая может содержать или не содержать 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, если есть, other, если нет

Класс java.util.Random

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

Регулярные выражения

- 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");
```