Stream API

Способы создания стримов:

1. Классический: Создание стрима из коллекции - collection.**stream**()

Collection<String> collection = Arrays.asList("a1", "a2", "a3");

Stream<String> streamFromCollection = collection.stream();

2. Создание стрима из значений - **Stream.of**(*значение1*,… *значениеN*)

Stream<String> streamFromValues = Stream.of("a1", "a2", "a3");

3. Создание стрима из массива - **Arrays.stream**(*массив*)

String[] array = {"a1","a2","a3"};

Stream<String> streamFromArrays = Arrays.stream(array);

4. Создание стрима из файла (каждая строка в файле будет отдельным элементом в стриме) - **Files.lines**(*путь\_к\_файлу*)

Stream<String> streamFromFiles = Files.lines(Paths.get("file.txt"))

5. Создание стрима из строки - «строка».**chars**()

IntStream streamFromString = "123".chars()

6. С помощью Stream.builder - Stream.**builder**().**add**(...)....**build**()

Stream.builder().add("a1").add("a2").add("a3").build()

7. Создание параллельного стрима - collection.**parallelStream**()

Stream<String> stream = collection.parallelStream();

8. Создание бесконечных стрима с помощью Stream.iterate - **Stream.iterate**(*начальное\_условие*, *выражение\_генерации*)

Stream<Integer> streamFromIterate = Stream.iterate(1, n -> n + 1)

9. Создание бесконечных стрима с помощью Stream.generate - **Stream.generate**(*выражение\_генерации*)

Stream<String> streamFromGenerate = Stream.generate(() -> "a1")

Example:

**public class** StreamDemo {  
 **public static void** main(String[] args) {  
 List<String> myList = Arrays.*asList*(**"a1"**, **"a2"**, **"b1"**, **"c2"**, **"c1"**);  
 myList  
 .stream() // открывает поток для выполнения  
 .filter(s -> s.startsWith(**"c"**)) // фильтр  
 .map(String::toUpperCase)  
 .sorted() // сортировка  
 .forEach(System.***out***::println);  
 }  
}

Описание методов Stream:

.filter(s -> s.startsWith(**"c"**)) - Отфильтровывает записи, возвращает только записи, соответствующие условию.

условия:

(s -> s.startsWith(**"c"**)) – делает поиск значений начинающихся с символа «С». Поиск можно задавать из нескольких символов.

(s -> s.startsWith(**"c", 1**)) – делает такой же поиск, цифра после запятой указывает с какой позиции начинается поиск.

(s -> s.equals(**"a1"**)) – ищет точное совпадение.

(s -> s.equalsIgnoreCase(**"A1"**)) – ищет совпадения игнорируя регистр.

(s -> s.contains(**"re"**)) – ищет совпадения символов, совпадения могут быть в любом месте строки.

myList.stream().filter(**"a1"**::equals).count() – ищет совпадения и возвращает количество совпадений.

.count() - возвращает количество элементов в стриме.

.skip(2) - Позволяет пропустить N первых элементов

myList.stream().skip(2).forEach(System.***out***::print); - пропускает первых два элемента.

.distinct() - возвращает стрим без дубликатов

myList.stream().distinct().forEach(System.***out***::print);

.map() - преобразует каждый элемент стрима

myList.stream().map(s -> s+**","**).collect(Collectors.*toList*()).forEach(System.***out***::println);

добавляется к каждому элементу запятую.

.collect() - представление результатов в виде коллекций и других структур данных

myList.stream().distinct().collect(Collectors.*toList*()).forEach(System.***out***::println);

.map() - Преобразует каждый элемент стрима

myList.stream().map(String::toUpperCase).forEach(System.***out***::println); - приводит каждый элемент в верхний регистр.

.peek() - возвращает тот же стрим, но применяет функцию к каждому элементу стрима

myList.stream().peek(e -> System.***out***.println(e+**","**)).collect(Collectors.*toList*()); - добавляет запятую к каждому элементу.

.limit(N) – позволяет ограничить выборку определенным количеством первых элементов

myList.stream().limit(3).collect(Collectors.*toList*()).forEach(System.***out***::println); - выводит первые три элемента.

.sorted() - позволяет сортировать значения либо в натуральном порядке, либо задавая Comparator

myList.stream().sorted().forEach(System.***out***::println); - сортирует значения

myList.stream().sorted(Comparator.*reverseOrder*()).forEach(System.***out***::println); - выводит отсортированные значения в обратном порядке.

.mapToInt()

.mapToDouble()   
.mapToLong() - аналог map, но возвращает числовой стрим (то есть стрим из числовых примитивов)

myList.stream().mapToInt(e -> Integer.*parseInt*(e)+1).forEach(System.***out***::println); - конвертирует элементы в числовой формат и прибавляет единицу.

System.***out***.println(myList.stream().findFirst().orElse(**"0"**));

.findFirst() - возвращает первый элемент из стрима (возвращает Optional)

.findAny() - возвращает любой подходящий элемент из стрима (возвращает Optional)

.anyMatch() - Возвращает true, если условие выполняется хотя бы для одного элемента

.noneMatch() - Возвращает true, если условие не выполняется ни для одного элемента

.allMatch() - Возвращает true, если условие выполняется для всех элементов

System.***out***.println(myList.stream().anyMatch(**"a2"**::equals));

System.***out***.println(myList.stream().allMatch(e -> e.contains(**"1"**)));

.min() – Возвращает минимальный элемент, в качестве условия использует компаратор

.max() - Возвращает максимальный элемент, в качестве условия использует компаратор

System.***out***.println(myList.stream().min(String::compareTo));

.forEach() - применяет функцию к каждому объекту стрима, порядок при параллельном выполнении не гарантируется

.forEachOrdered() - применяет функцию к каждому объекту стрима, сохранение порядка элементов гарантирует

myList.stream().forEachOrdered(System.***out***::println);

.toArray() - Возвращает массив значений стрима

List<String> myList = Arrays.*asList*(**"1"**,**"b1"**,**"ee1w2"**,**"a1"**,**"s1"**,**"s12s"**);  
String[] list = myList.stream().map(String::toUpperCase).toArray(String[]::**new**);  
Arrays.*stream*(list).forEach(System.***out***::println);