Java Pro 5.2

TEL-RAN
by Starta Institute

Optional. Stream API

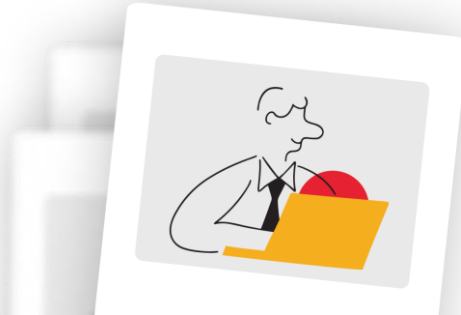
ПРЕПОДАВАТЕЛЬ

TEL-RAN
by Starta Institute

Юрий Костяной

Java/Kotlin backend-разработчик

- 3+ года опыта в коммерческой разработке
- 2+ года опыта в преподавании
- Проекты по интеграции сторонних платформ, CRM
- Проблемно-ориентированный подход в преподавании



ВАЖНО:

- Камера должна быть включена на протяжении всего занятия.
- Если у Вас возник вопрос в процессе занятия, пожалуйста, поднимите руку и дождитесь, пока преподаватель закончит мысль и спросит Вас, также можно задать вопрос в чате или когда преподаватель скажет, что начался блок вопросов.
- Организационные вопросы по обучению решаются с кураторами, а не на тематических занятиях.
- Вести себя уважительно и этично по отношению к остальным участникам занятия.
- Во время занятия будут интерактивные задания, будьте готовы включить камеру или демонстрацию экрана по просьбе преподавателя.

TEL-RAN
by Starta Institute

ПЛАН ЗАНЯТИЯ

1. Повторение
2. Основной блок
3. Вопросы по основному блоку
4. Домашняя работа

TEL-RAN
by Starta Institute

1 ПОВТОРЕНИЕ ИЗУЧЕННОГО

Повторение

TEL-RAN
by Starta Institute

Что такое лямбда-выражение? Какова его структура?

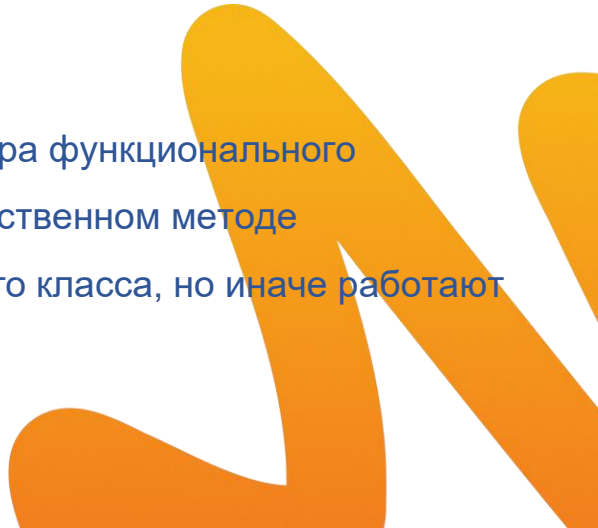
Повторение

TEL-RAN
by Starta Institute

Что такое лямбда-выражение? Какова его структура?

Лямбда выражение – это короткая форма записи создания экземпляра функционального интерфейса, позволяющее определить, что будет выполнено в единственном методе функционального интерфейса. Они аналогичны созданию анонимного класса, но иначе работают с ключевым словом this.

Структура лямбда-выражения: (аргументы) -> {тело}.



Повторение

TEL-RAN
by Starta Institute

Что будет выведено в консоль?

```
List<String> names = new ArrayList<>(List.of("Alice", "Bob", "Charlie"));
names.removeIf(e -> e.toLowerCase().startsWith("b"));
names.forEach(name -> System.out.print(name + " "));
```

Повторение

TEL-RAN
by Starta Institute

Что будет выведено в консоль?

```
List<String> names = new ArrayList<>(List.of("Alice", "Bob", "Charlie"));
names.removeIf(e -> e.toLowerCase().startsWith("b"));
names.forEach(name -> System.out.print(name + " "));
```

Alice Charlie

Повторение

TEL-RAN
by Starta Institute

Какие функциональные интерфейсы нужно указать?

```
XYZ square = x -> x * x;
XYZ isEven = x -> x % 2 == 0;
XYZ parseToInt = Integer::parseInt;
XYZ greeting = () -> "Hello, world!";
XYZ increment = x -> x + 1;
XYZ parseMe = s -> {
    try {
        return Integer.parseInt(s);
    } catch (NumberFormatException e) {
        return -1;
    }
};
XYZ add = Integer::sum;
```

Повторение

TEL-RAN
by Starta Institute

Какие функциональные интерфейсы нужно указать?

```
Function<Integer, Integer> square = x -> x * x;
Predicate<Integer> isEven = x -> x % 2 == 0;
Function<String, Integer> parseToInt = Integer::parseInt;
Supplier<String> greeting = () -> "Hello, world!";
UnaryOperator<Integer> increment = x -> x + 1;
Function<String, Integer> parseMe = s -> {
    try {
        return Integer.parseInt(s);
    } catch (NumberFormatException e) {
        return -1;
    }
};
BiFunction<Integer, Integer, Integer> add = Integer::sum;
```

Повторение

TEL-RAN
by Starta Institute

Что будет выведено в консоль?

```
System.out.println(square.apply(5));
System.out.println(isEven.test(7));
System.out.println(parseToInt.apply("42"));
System.out.println(greeting.get());
System.out.println(increment.apply(7));
System.out.println(parseMe.apply("42"));
System.out.println(add.apply(3, 4));
```

```
Function<Integer, Integer> square = x -> x * x;
Predicate<Integer> isEven = x -> x % 2 == 0;
Function<String, Integer> parseToInt = Integer::parseInt;
Supplier<String> greeting = () -> "Hello, world!";
UnaryOperator<Integer> increment = x -> x + 1;
Function<String, Integer> parseMe = s -> {
    try {
        return Integer.parseInt(s);
    } catch (NumberFormatException e) {
        return -1;
    }
};
BiFunction<Integer, Integer, Integer> add = Integer::sum;
```

Повторение

Что будет выведено в консоль?

Вывод:

25

false

42

Hello, world!

8

42

7

```
Function<Integer, Integer> square = x -> x * x;
Predicate<Integer> isEven = x -> x % 2 == 0;
Function<String, Integer> parseToInt = Integer::parseInt;
Supplier<String> greeting = () -> "Hello, world!";
UnaryOperator<Integer> increment = x -> x + 1;
Function<String, Integer> parseMe = s -> {
    try {
        return Integer.parseInt(s);
    } catch (NumberFormatException e) {
        return -1;
    }
};
BiFunction<Integer, Integer, Integer> add = Integer::sum;

System.out.println(square.apply(5));
System.out.println(isEven.test(7));
System.out.println(parseToInt.apply("42"));
System.out.println(greeting.get());
System.out.println(increment.apply(7));
System.out.println(parseMe.apply("42"));
System.out.println(add.apply(3, 4));
```

Повторение

TEL-RAN
by Starta Institute

Исправьте ошибку в коде

```
public class Main {
    public static void main(String[] args) {
        WorkerInterface w = () -> System.out.println("I'm working");
        w.doSomeWork();
    }

    @FunctionalInterface
    public interface WorkerInterface {
        void doSomeWork();
        void doAnotherWork();
    }
}
```

Повторение

TEL-RAN
by Starta Institute

Исправьте ошибку в коде

```
public class Main {
    public static void main(String[] args) {
        WorkerInterface w = () -> System.out.println("I'm working");
        w.doSomeWork();
    }

    @FunctionalInterface
    public interface WorkerInterface {
        void doSomeWork();
    }
}
```

В функциональном интерфейсе должен быть только один абстрактный метод.

Повторение

TEL-RAN
by Starta Institute

В чём прикол мема?

TEL-RAN
by Starta Institute

2 ОСНОВНОЙ БЛОК

Введение

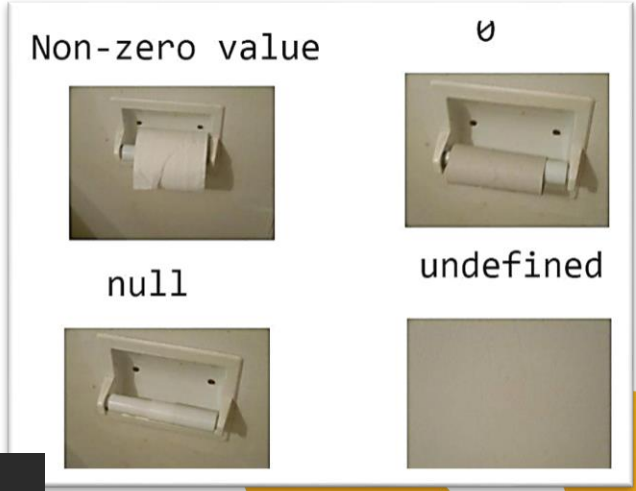
TEL-RAN
by Starta Institute

- По-null-ам
- Конвейер

Проблема

TEL-RAN
by Starta Institute

Переменные ссылочного типы, как мы знаем могут хранить разные значения и в том числе *null*, т.е. ссылку «в никуда». Иногда дополнительное состояние в виде *null* даёт нам дополнительную возможность хранить информацию, что значение не было получено по какой-то причине.




```
public static void main(String[] args) {
    System.out.println(getGreaterThanThat(10, List.of(0, 1, 2, 3, 4, 5)));
}
private static Integer getGreaterThanThat(int that, List<Integer> nums) {
    Integer num = null;
    for (int i : nums) { if (i > that) num = i; }
    return num;
}
```

Проблема

Но гораздо чаще это приводит к появлению *NullPointerException* (NPE). NPE возникает, когда в переменной лежит *null*, но мы пытаемся вызвать у переменной метод, соответствующий её типу. Например,

```
String str = null;
System.out.println(str.length());
```

С NPE сталкиваются как начинающие, так и опытные разработчики.

Неужели нет способа сократить количество таких исключений?

Проблема

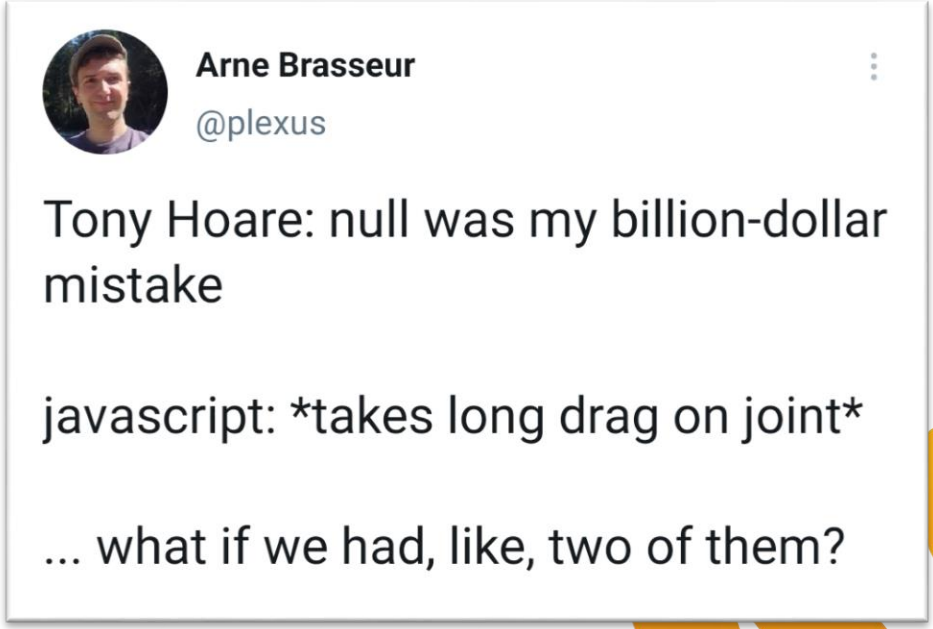
TEL-RAN
by Starta Institute

В Java ещё не всё так плохо.

«Изобретение null в 1965 году было мой ошибкой на миллиард долларов. Я не мог устоять перед искушением ввести нулевую ссылку просто потому, что ее было так легко реализовать»

Но неужели нет способа сократить количество таких исключений?

Просто всё нужно проверять на null.



По-null-ам

Optional

TEL-RAN
by Starta Institute

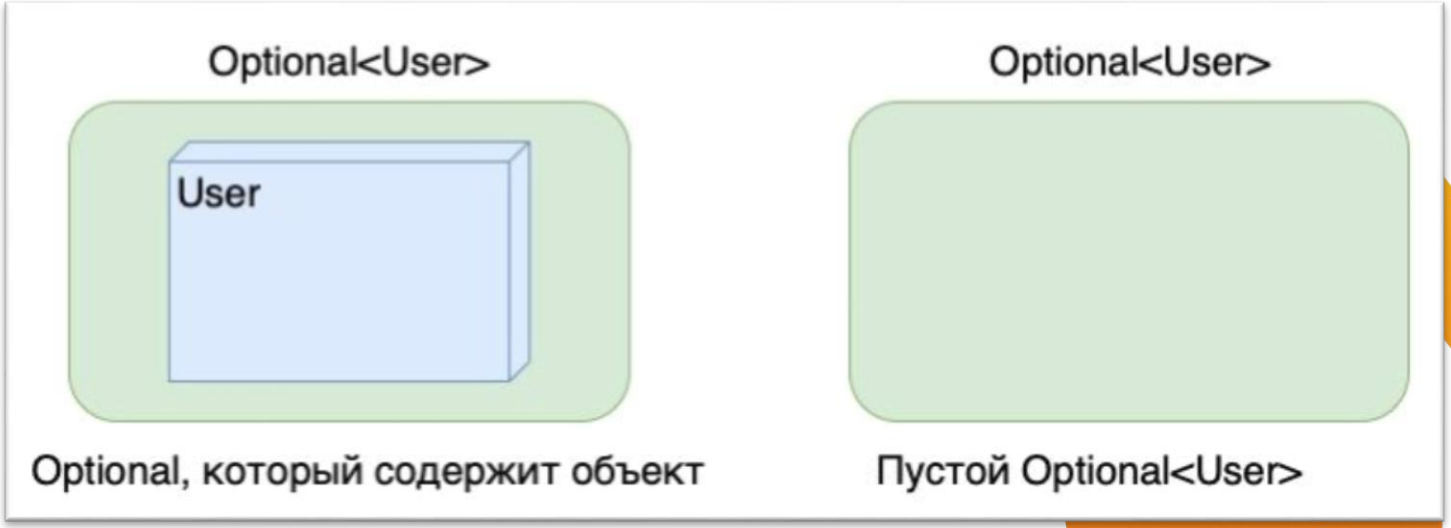
Для того, чтобы не писать длинный код и автоматизировать проверку на null в Java 8 придумали класс **Optional<T>** из пакета java.util, который является параметризируемой null-safe обёрткой.

По-null-ам

Optional

TEL-RAN
by Starta Institute

Если мы ищем объект User в мапе по его ключу, то может оказаться, что такого ключа в мапе нет и мы получим *null*. Вместо того, чтобы передавать *null* мы можем завернуть результат в Optional. Такой объект не будет null и мы сможем не опасаться NPE.



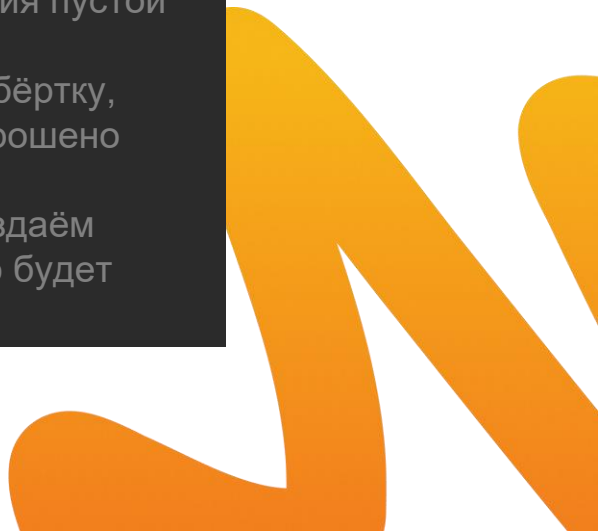
По-null-ам

Создание Optional

TEL-RAN
by Starta Institute

У Optional закрытый конструктор. Получать новые экземпляры Optional можно одним из следующих методов:

```
public static <T> Optional<T> empty() // метод создания пустой обёртки
public static <T> Optional<T> of(T value) // создаём обёртку, помещая в неё value. Если value == null, будет выброшено NPE (ранний отлов null-значений)
public static <T> Optional<T> ofNullable(T value) // создаём обёртку, помещая в неё value. Если value == null, то будет создана пустая обёртка
```



По-null-ам

Методы Optional

TEL-RAN
by Starta Institute

```
public T get(); // получение значения из обёртки
public T orElseGet(Supplier<? extends T> supplier); // получить значение или, если обёртка пуста, то получить его из лямбда-выражения
public T orElseThrow(); // получить значение или, если обёртка пуста, бросить NoSuchElementException
public <X extends Throwable> T orElseThrow(Supplier<? extends X> exceptionSupplier) throws X; // получить значение или, если обёртка пуста, бросить исключение, полученное из лямбда-выражения
public boolean isPresent(); // проверка, что обёртка не пуста
public boolean isEmpty(); // проверка, что обёртка пуста
public void ifPresent(Consumer<? super T> action); // проверка, что объект представлен в функциональном стиле.
//Если объект не empty, то будет выполнено лямбда-выражение с представленным значением.
public void ifPresentOrElse(Consumer<? super T> action, Runnable emptyAction); // то же, но в случае пустой обёртке выполняется emptyAction
```

Метды *toString*, *equals* и *hashCode* в Optional переопределены.

По-null-ам

Пример Optional

TEL-RAN
by Starta Institute

```
public static void main(String[] args) {
    Optional<Integer> num = getGreaterThanThat(10, List.of(0, 1, 2, 3, 4, 5));
    if (num.isPresent()) { // проверка, что значение представлено (не null)
        System.out.println(num.get()); // чтобы получить значение из обёртки, вызываем get()
    }
    if (num.isEmpty()) System.out.println(num); // обратная проверка
    num.ifPresent(System.out::println); // проверка, что объект представлен в функциональном стиле.
}

private static Optional<Integer> getGreaterThanThat(int that, List<Integer> nums) {
    Optional<Integer> num = Optional.empty();
    for (int i : nums) {
        if (i > that) num = Optional.of(i);
    }
    return num;
}
```

Задание

TEL-RAN
by Starta Institute

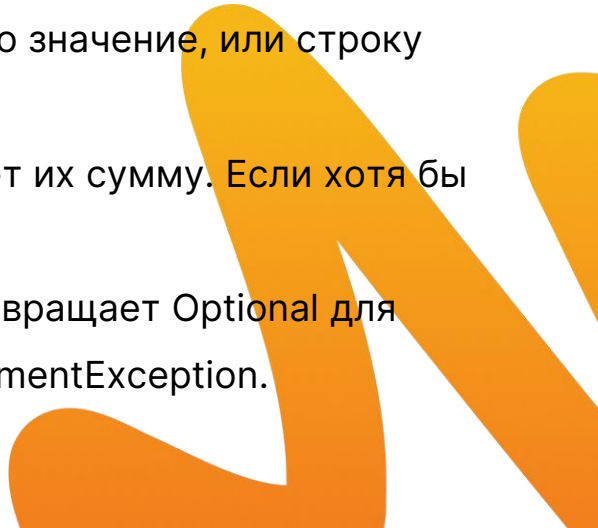
1 Создайте метод, который принимает объект и возвращает его строковое представление. Используйте Optional для предотвращения NullPointerException, если переданный объект равен null.

2 Создайте метод, который возвращает Optional для строки. Если строка начинается с буквы "A", верните её значение, иначе верните Optional.empty().

3 Создайте метод, который принимает Optional и возвращает его значение, или строку "Default", если значение отсутствует.

4 Создайте метод, который принимает два Optional и возвращает их сумму. Если хотя бы одно из значений отсутствует, верните Optional.empty().

5 Создайте метод, который принимает Optional для строки и возвращает Optional для длины этой строки. Если входная строка null, бросьте IllegalArgumentException.



По-null-ам

Применение Optional

TEL-RAN
by Starta Institute

Optional в первую очередь предназначен для использования в качестве типа возвращаемого значения метода, когда существует явная необходимость представлять «отсутствие результата» и где использование null может вызвать ошибки.

По-null-ам

Применение Optional

TEL-RAN
by Starta Institute

Не стоит применять *Optional* в качестве:

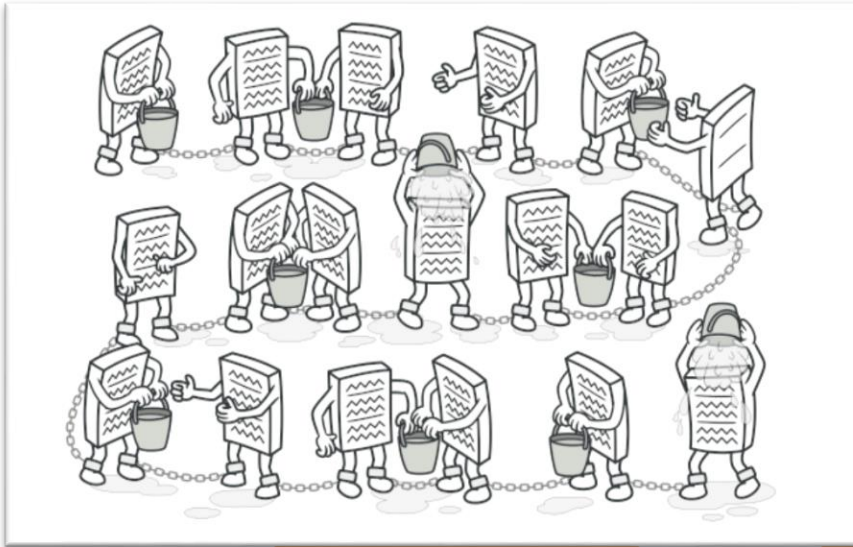
- параметра метода. Если пользователь метода с параметром *Optional* не знает об этом, он может передать методу *null* вместо *Optional.empty()*. И обработка *null* приведет к исключению *NullPointerException*.
- для объявления поля класса. Могут возникнуть проблемы с некоторыми фремворками, как *Spring Data/Hibernate*. Хотя некоторые фреймворки, такие как *Jackson* отлично интегрируют Optional в свою экосиситему. Ещё *Optional* не имплементирует интерфейс *Serializable*, т.е. поля не будут сериализоваться.

По-null-ам

Цепочка обязанностей

TEL-RAN
by Starta Institute

Паттерн Цепочка обязанностей https://refactoring.guru/ru/design-patterns/chain-of-responsibility, суть которого сводится в последовательной обработке запроса, пришедшего в программу, а также ранее изученный нами паттерн Команда https://refactoring.guru/ru/design-patterns/command, который является основой использования лямбда-выражений, натолкнули разработчиков Java на мысль, что можно создать конвейерных подход обработки некоторого значения, что сильно сокращает код, т.к. позволяет делать цепочку вызовов, а не отдельные инструкции.



По-null-ам

Потоковые методы Optional

TEL-RAN
by Starta Institute

```
public Optional<T> filter(Predicate<? super T> predicate) // метод для фильтрации содержимого optional по заданному предикату
public <U> Optional<U> map(Function<? super T, ? extends U> mapper) // метод для замены значения с поммощью переданного лямбда-выражения (обычно на основании текущего значения)
public <U> Optional<U> flatMap(Function<? super T, ? extends Optional<? extends U>> mapper) // метод аналогичен map, но результат переданного лямбда-выражения уже является Optional
public Optional<T> or(Supplier<? extends Optional<? extends T>> supplier) // взять текущее значение, но если обёртка пуста, то выполнить лямбда-выражение
public Stream<T> stream() // если значение представлено, то получить поток (конвейер) из одно элемента - представленного значения
```

По-null-ам

Потоковые методы Optional

TEL-RAN
by Starta Institute

Потоковые методы созданы так, чтобы делать цепочку вызовов. Они отлично комбинируются.

```
private static DayOfWeek getWeekend(LocalDate date) {
    DayOfWeek dayWeek = Optional.of(date)
            .map(LocalDate::getDayOfWeek)
            .filter(DayOfWeek.SUNDAY::equals)
            .orElse(DayOfWeek.SATURDAY);
    return dayWeek;
}
```

Задание

Используйте потоковые функции:

1 Создайте метод, который принимает Optional для строки и возвращает Optional для длины этой строки.

2 Создайте метод, который принимает Optional и возвращает true, если значение присутствует и является положительным числом, иначе false.

3 Создайте метод, который принимает два списка строк и возвращает их объединение. Сначала метод берёт очередное значение из первой очереди. Если оно null, то берёт значение из второго списка. Если значение второго списка null, то берётся значение по умолчанию.

Проблема

TEL-RAN
by Starta Institute

Все операции над коллекциями и массивами можно собрать в некоторый набор по типам:

- Перебрать все элементы (например, чтобы посчитать)
- Отфильтровать нужные/не нужные элементы
- Найти заданный максимальный/минимальный элемент
- Отсортировать коллекцию нужным образом
- Взять каждый элемент и что-то с ним сделать (например вывести в консоль)
- Взять каждый элемент и изменить его
- Собрать все элементы в новую коллекцию и тд.

Можем ли применить подход конвейерной обработки для решения этих задач?

Конвейер

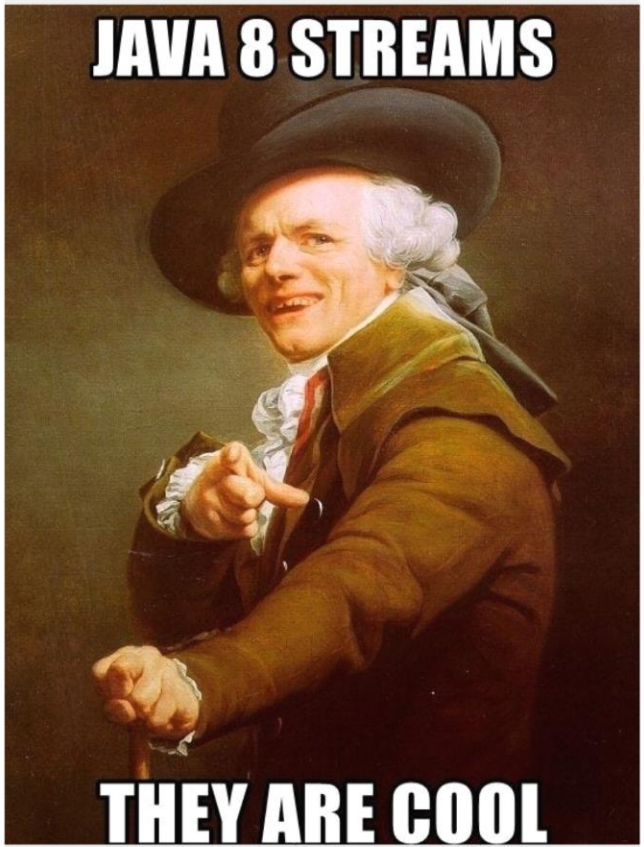
Stream API

TEL-RAN
by Starta Institute

Stream API – способ работать со структурами данных в функциональном стиле, появившийся в Java 8. Этот способ позволил писать значительно более короткий код.

Stream (поток, не путайте с потоками ввода вывода и потоками многопоточной программы *Thread*) – это основной класс конвейерного подхода. Он параметризирован и хранится в пакете *java.util.stream*.

```
List<String> list = new ArrayList<>();
list.add("One");
list.add("Two");
list.add("Three");
Stream<String> stream = list.stream();
```



Конвейер
Stream

TEL-RAN
by Starta Institute

Stream является наследником класса BaseStream. У данного класса есть и другие наследники, следующие концепции конвейерной обработки. Например, *IntStream.*

```
// До Stream API
int[] arr = {50, 60, 70, 80, 90, 100, 110, 120};
int count = 0;
for (int x : arr) {
    if (x >= 90) continue;
    x += 10;
    count++;
    if (count > 3) break;
    System.out.print(x);
}

// После Stream API
IntStream.of(50, 60, 70, 80, 90, 100, 110, 120).filter(x -> x < 90).map(x -> x + 10)
        .limit(3).forEach(System.out::print);
```

Конвейер

Создание Stream

TEL-RAN by Starta Institute

Потоком может стать любой набор однотипных данных. Основные способы создания потоков:

- Пустой стрим: *Stream.empty()*
- Стрим из Optional: *op.stream()* (поток из одного элемента, взятого из обёртки)
- Стрим из List: *list.stream()* (поток из всех элементов списка)
- Стрим из Map: *map.entrySet()*.stream() (поток из всех пар ключ-значения мапы)
- Стрим из массива: *Arrays.stream(array)* (поток из жлементов массива)
- Стрим из указанных элементов: *Stream.of*("1", "2", "3")
- Стрим из указанных элементов: *Stream.ofNullable*(elem) (поток из одного элемента, если он не null, в противном случае – пустой поток)
- Стрим из стримов: concat(stream1, stream2) (новый поток, состоящий из элементов первого потока и следующих за ними элементов второго потока).

Конвейер

Методы Stream

TEL-RAN
by Starta Institute

Все методы стримов можно разделить на промежуточные и терминальные.

Промежуточные (“intermediate”, ещё называют “lazy”) – это те, в результате которых получается новый стрим, т.е. они берут элемент из входного потока, что-то с ним делают и кладут результат в выходной поток. Таких методов в стриме может быть много.

Терминальные (“terminal”, ещё называют “eager”) – методы, которые стоят в конце потока и завершают его. В потоке может быть только один терминальный метод. Обычно такие методы возвращают результат, отличный от Stream. Обработка не начнётся до тех пор, пока не будет вызван терминальный оператор.

Конвейер

Промежуточные методы Stream

TEL-RAN by Starta Institute

filter	отработает как фильтр, вернет значения, которые подходят под заданное условие	collection.stream() .filter(«e22»::equals).count()
sorted	отсортирует элементы в естественном порядке; можно использовать Comparator	collection.stream() .sorted().collect(Collectors.toList())
limit	лимитирует вывод по тому, количеству, которое вы укажете	collection.stream() .limit(10).collect(Collectors.toList())
skip	пропустит указанное вами количество элементов	collection.stream() .skip(3).findFirst().orElse("4")
distinct	найдет и уберет элементы, которые повторяются; вернет элементы без повторов	collection.stream() .distinct().collect(Collectors.toList())
peek	выполнить действие над каждым элементом элементов, вернет стрим с исходными элементами	collection.stream() .map(String::toLowerCase).peek((e) -> System.out.print("," + e)) . collect(Collectors.toList())

Конвейер

Промежуточные методы Stream

map	выполнит действия над каждым элементом; вернет элементы с результатами функций	Stream.of("3", "4", "5") .map(Integer::parseInt) .map(x -> x + 10) .forEach(System.out::println);
mapToInt, mapToDouble, mapToLong	Сработает как map, только вернет числовой stream	collection.stream() .mapToInt((s) -> Integer.parseInt(s)).toArray()
flatMap, flatMapToInt, flatMapToDouble, flatMapToLong	сработает как map, но преобразует один элемент в множество других (новый поток)	collection.stream() .flatMap((p) -> Arrays.asList(p.split(",")).stream()).toArray(String[]::)

Конвейер

Терминальные методы Stream

findFirst	вернет элемент, соответствующий условию, который стоит первым	collection.stream() .findFirst().orElse("10")
findAny	вернет любой элемент, соответствующий условию	collection.stream() .findAny().orElse("10")
collect	соберет результаты обработки в коллекции и не только	collection.stream() .filter((s) -> s.contains("10")) .collect(Collectors.toList())
count	посчитает и выведет, сколько элементов, соответствующих условию	collection.stream() .filter("f5"::equals).count()
anyMatch	**true**, когда хоть один элемент соответствует условиям	collection.stream() .anyMatch("f5"::equals)
noneMatch	**true**, когда ни один элемент не соответствует условиям	collection.stream() .noneMatch("b6"::equals)
allMatch	**true**, когда все элементы соответствуют условиям	collection.stream() .allMatch((s) -> s.contains("8"))

Конвейер

Терминальные методы Stream

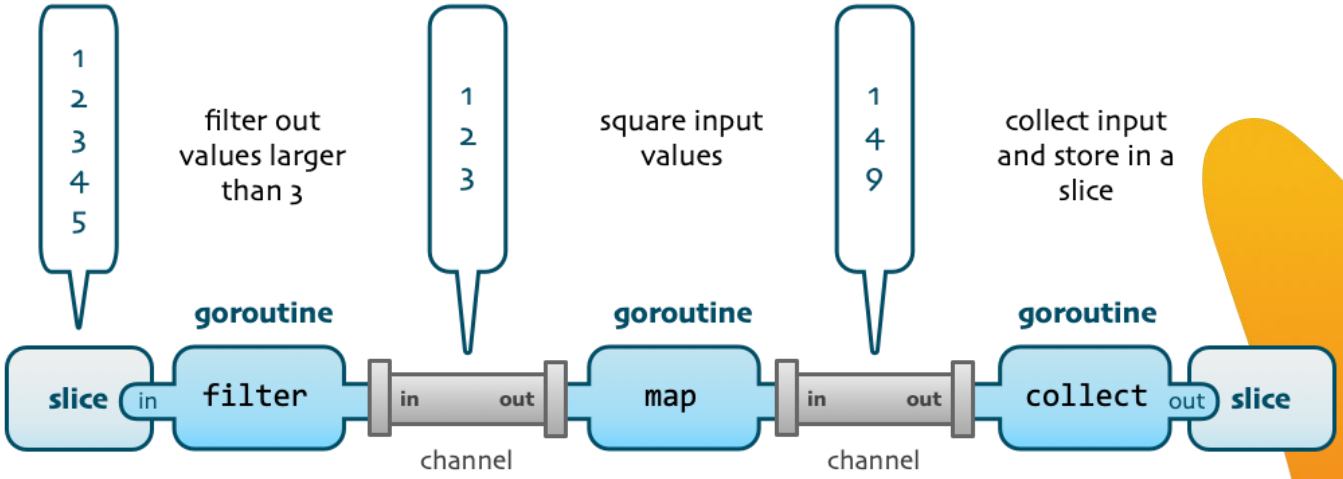
TEL-RAN by Starta Institute

min	найдет самый маленький элемент, используя переданный сравнитель	collection.stream() .min(String::compareTo).get()
max	найдет самый большой элемент, используя переданный сравнитель	collection.stream() .max(String::compareTo).get()
forEach	применит функцию ко всем элементам, но порядок выполнения гарантировать не может	set.stream() .forEach((p) -> p.append("_2"));
forEachOrdered	применит функцию ко всем элементам по очереди, порядок выполнения гарантировать может	list.stream() .forEachOrdered((p) -> p.append("_nv"));
toArray	приведет значения стрима к массиву	collection.stream() .map(String::toLowerCase).toArray(String[]::**new**);
reduce	преобразует все элементы в один объект	collection.stream() .reduce((c1, c2) -> c1 + c2).orElse(0)

Конвейер
Stream

TEL-RAN
by Starta Institute

Stream можно представить конвейером, применяющим указанные методы.



Конвейер

Метод collect

toList - стрим приводится к списку;

toCollection - получаем коллекцию;

toSet - получаем множество;

toConcurrentMap, toMap - получаем map;

summingInt, summingDouble, summingLong - если требуется получить сумму чисел;

averagingInt, averagingDouble, averagingLong - если хотите вернуть среднее значение;

groupingBy - если необходимо разбить коллекцию на части. toList - стрим приводится к списку;

Конвейер

Методы числовых стримов

TEL-RAN
by Starta Institute

sum	вернет сумму чисел, представленных в коллекции	collection.stream() .mapToInt((s) -> Integer.parseInt(s)) .sum()
average	вернет среднее арифметическое	collection.stream() .mapToInt((s) -> Integer.parseInt(s)) .average()
mapToObj	преобразует числовой стрим в объектный	intStream .mapToObj((id) -> **new** Key(id)).toArray()

Конвейер

Методы параллельных стримов

TEL-RAN by Starta Institute

isParallel	скажет, параллельный стрим или нет	someStream.isParallel()
parallel	сделает стрим параллельным или вернет сам себя	someStream = stream.parallel()
sequential	сделает стрим последовательным или вернет сам себя	someStream = stream.sequential()

Стримы могут быть последовательными и параллельными. Первые выполняются в текущем потоке. В параллельном стриме элементы разделяются на группы. Их обработка проходит в каждом потоке по отдельности. Затем они снова объединяются, чтобы вывести результат. С помощью методов parallel и sequential

Не рекомендуется применять параллельность для выполнения долгих операций (например, извлечения данных из базы. Долгие операции могут остановить работу всех параллельных стримов в Java Virtual Machine.

Задание

TEL-RAN
by Starta Institute

1 Создайте список чисел и используйте Stream API, чтобы отфильтровать только четные числа.

2 Создайте список строк и используйте Stream API, чтобы создать новый список, где каждая строка будет в верхнем регистре.

3 Создайте список строк и используйте Stream API для сортировки строк в алфавитном порядке.

4 Создайте список чисел и используйте Stream API, чтобы получить первые три элемента, а затем пропустить следующие два.

5 Создайте список строк и используйте Stream API, чтобы найти первую строку, начинающуюся с определенной буквы.

6 Создайте список объектов с числовым полем и используйте Stream API для вычисления суммы этих числовых полей.

Задание

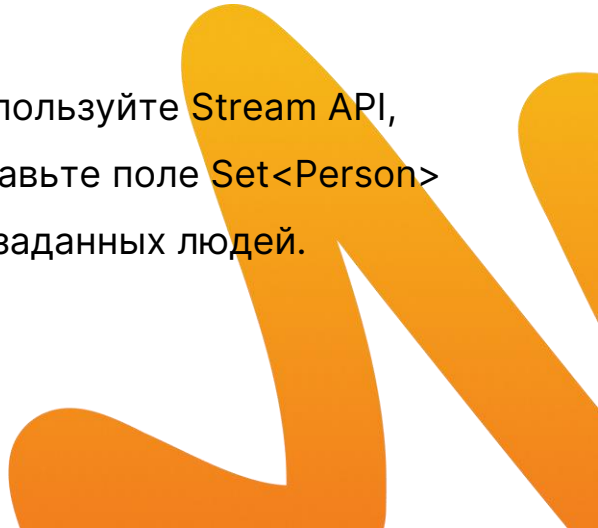
TEL-RAN
by Starta Institute

7 Создайте список объектов с полем категории и используйте Stream API, чтобы сгруппировать объекты по их категориям.

8 Создайте список строк и используйте Stream API, чтобы объединить все строки в одну строку, разделяя их запятой.

9 Создайте список чисел и используйте Stream API, чтобы проверить, все ли числа положительные.

10 Создайте список объектов Person с полями "id" и "name" и используйте Stream API, чтобы создать Map, где ключ - это "id", а значение - "name". Добавьте поле Set<Person> friends. Создайте метод получения «общего круга друзей» всех заданных людей.



TEL-RAN
by Starta Institute

3 Домашнее задание

Домашнее задание

TEL-RAN
by Starta Institute

1.1 Создайте класс Дом, содержащий список Квартир. Каждая квартира содержит список комнат. Каждая комната содержит площадь. С помощью Stream API посчитайте суммарную жилую площадь дома.

1.2 Среди всех квартир найдите квартиру, площадь которой больше 100 кв. м. Если такая квартира найдена, то выведите её площадь. В противном случае выведите сообщение, что такой квартиры нет.

1.3 Старый дом расселяют. В новом доме жильцы должны получить квартиры большей площади. Создайте новый список квартир дома, увеличив площадь каждой комнаты на 30%.

1.4 Дан список, каждый элемент которого – это мапа площадей комнат (комната - площадь). Создайте новые квартиры с комнатами заданных площадей.

ЗАКЛЮЧЕНИЕ

TEL-RAN
by Starta Institute