Java Pro 5.1

TEL-RAN
by Starta Institute

Лямбда-выражения. Функциональные интерфейсы

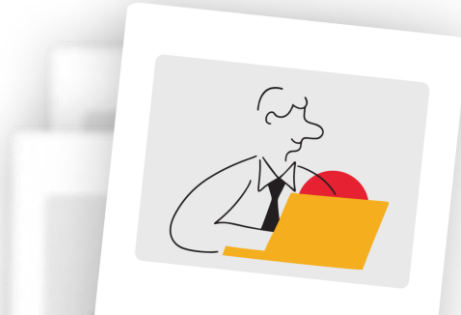
ПРЕПОДАВАТЕЛЬ

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

На каком уровне модели OSI применяются протоколы TCP, IP, HTTP?

Повторение

TEL-RAN
by Starta Institute

На каком уровне модели OSI применяются протоколы TCP, IP, HTTP?

HTTP – прикладной уровень

TCP – транспортный уровень

IP – сетевой уровень

Повторение

TEL-RAN
by Starta Institute

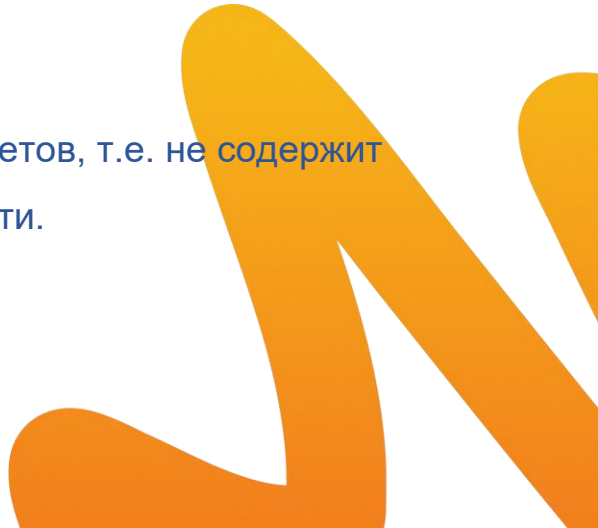
В чем основное отличие TCP от UDP?

Повторение

TEL-RAN
by Starta Institute

В чем основное отличие TCP от UDP?

UDP, в отличие от TCP, не гарантирует подтверждения доставки пакетов, т.е. не содержит механизма повторной отправки при потере данных из-за проблем сети.



Повторение

TEL-RAN
by Starta Institute

Какие классы помогают организовать обмен данными между клиентом и сервером?

Повторение

TEL-RAN
by Starta Institute

Какие классы помогают организовать обмен данными между клиентом и сервером?

Socket и ServerSocket

Повторение

TEL-RAN
by Starta Institute

Что помогает клиентскому приложению найти сервер в сети?

Повторение

TEL-RAN
by Starta Institute

Что помогает клиентскому приложению найти сервер в сети?

IP-адрес или host, а также порт

Повторение

TEL-RAN
by Starta Institute

Какой класс поможет скачать web-страницу из Интернета?

Повторение

TEL-RAN
by Starta Institute

Какой класс поможет скачать web-страницу из Интернета?

Класс URL

Повторение

TEL-RAN
by Starta Institute

Исправьте ошибку в коде

```
ServerSocket serverSocket = new ServerSocket();
serverSocket.accept();
```

Повторение

TEL-RAN
by Starta Institute

Исправьте ошибку в коде

```
ServerSocket serverSocket = new ServerSocket(8080);
serverSocket.accept();
```

При создании объекта ServerSocket необходимо указать порт в конструкторе, либо после конструктора вызвать метод bind().

Повторение

TEL-RAN
by Starta Institute

Исправьте ошибку в коде

```
public static void main(String[] args) {
    Socket socket = new Socket("example.com", 8080);
}
```

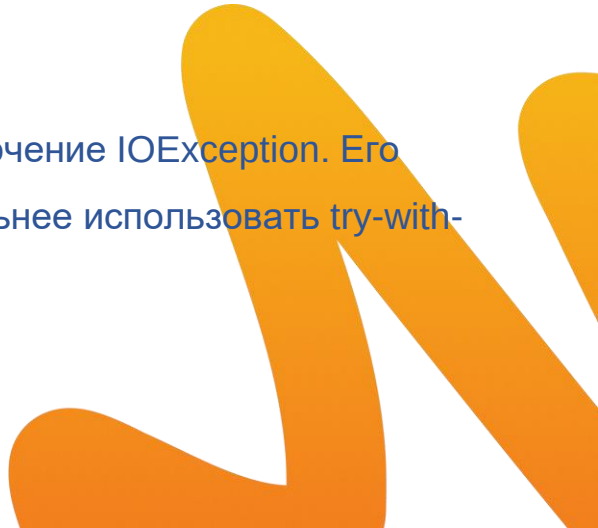
Повторение

TEL-RAN
by Starta Institute

Исправьте ошибку в коде

```
public static void main(String[] args) throws IOException {
    Socket socket = new Socket("example.com", 8080);
}
```

При создании объекта Socket может быть выброшено checked-исключение IOException. Его нужно либо указать в сигнатуре метода, либо отловить в try. Правильнее использовать try-with-resources, потому что сокеты и их потоки нужно закрывать.



Повторение

TEL-RAN
by Starta Institute

Исправьте ошибку в коде

```
public static void main(String[] args) throws IOException {
    ServerSocket serverSocket = new ServerSocket(8080);
    Socket socket = serverSocket.accept();
    // код использования сокета
    socket.close();
}
```

Повторение

TEL-RAN
by Starta Institute

Исправьте ошибку в коде

```
public static void main(String[] args) throws IOException {
    ServerSocket serverSocket = new ServerSocket(8080);
    Socket socket = serverSocket.accept();
    // код использования сокета
    socket.close();
    serverSocket.close();
}
```

При закрытии сокетов сервера нужно закрывать как объект Socket, так и объект ServerSocket.

Повторение

TEL-RAN
by Starta Institute

Исправьте ошибку в коде

```
public static void main(String[] args) throws IOException {
    InputStream inputStream1 = new URL("http://example.com").getInputStream();
}
```

Повторение

TEL-RAN
by Starta Institute

Исправьте ошибку в коде

```
public static void main(String[] args) throws IOException {
    InputStream inputStream1 = new URL("http://example.com").openConnection().getInputStream();
}
```

Пропущен метод openConnection(), получающий соединение.

Повторение

TEL-RAN
by Starta Institute

В чём прикол мема?

Повторение

TEL-RAN
by Starta Institute

Исправьте ошибку в коде

```
public static void main(String[] args) throws IOException {
    ObjectMapper objectMapper = new ObjectMapper();
    String jsonString = "{\"name\":\"John\",\"age\":30}";
    Person person = objectMapper.readValue(jsonString, Person.class);
}

public class Person {
    private String name;
    private int age;


    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

Повторение

TEL-RAN
by Starta Institute

Исправьте ошибку в коде

```
public static void main(String[] args) throws IOException {
    ObjectMapper objectMapper = new ObjectMapper();
    String jsonString = "{\"name\":\"John\",\"age\":30}";
    Person person = objectMapper.readValue(jsonString, Person.class);
}

public class Person {
    private String name;
    private int age;

    @JsonCreator
    public Person(@JsonProperty("name") String name, @JsonProperty("age") int age) {
        this.name = name;
        this.age = age;
    }
}
```

Необходимо удостовериться, что класс Person имеет конструктор по умолчанию (без аргументов), или добавить аннотацию @JsonCreator к конструктору, чтобы Jackson знал, как создавать экземпляры класса.

Повторение

TEL-RAN
by Starta Institute

Исправьте ошибку в коде

```
public class MainJson2 {
    ObjectMapper objectMapper = new ObjectMapper();
    Person person = new Person("Alice", 25);

    String jsonString = objectMapper.readTree(person);


    public class Person {
        private String name;
        private int age;

        public Person() { }
    }
}
```

Повторение

TEL-RAN
by Starta Institute

Исправьте ошибку в коде

Для сериализации объекта в JSON используется метод writeValueAsString, а не readTree.

```
public class MainJson2 {
    ObjectMapper objectMapper = new ObjectMapper();
    Person person = new Person("Alice", 25);

    String jsonString = objectMapper.writeValueAsString(person);


    public class Person {
        private String name;
        private int age;

        public Person() { }
    }
}
```

TEL-RAN
by Starta Institute

2 ОСНОВНОЙ БЛОК

Введение

TEL-RAN
by Starta Institute

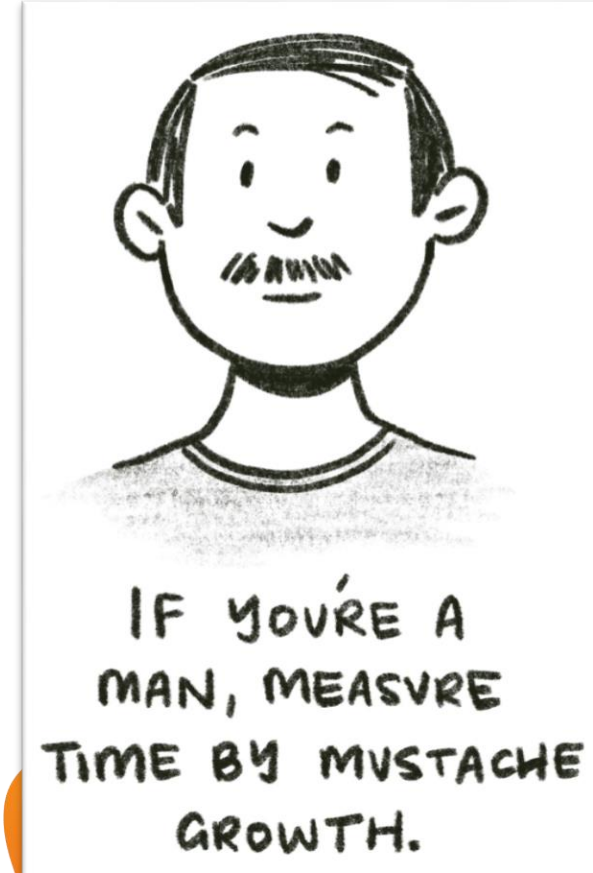
- Не командуй!
- Зовём и получаем
- И как это функционирует?

Проблема

TEL-RAN
by Starta Institute

Представьте, что Вы написали несколько сортировок и хотите измерить время выполнения каждой из них, чтобы понять, какая сортировка получилась эффективнее.
Для каждой сортировки нужно будет замерить время до запуска, затем запустить сортировку и измерить время после запуска. Придётся повторить код измерения времени для каждой сортировки. Это явное нарушение принципа *DRY*.

Вот бы написать метод, который принимал на вход сортировку, замерял время, запускал сортировку и затем снова замерял время по её окончанию. Но передавать методы в качестве аргументов в Java нельзя.



Не командуй!

Паттерн команда

TEL-RAN
by Starta Institute

Паттерн **Команда** предлагает решение этой проблемы.

https://refactoring.guru/ru/design-patterns/command

Излагая кратко, паттерн предлагает передавать не ссылки на методы, а объекты, которые хранят нужные методы (команды).

Передача объектов в аргументы – дело привычное для Java. Для того, чтобы реализовать это, достаточно создать интерфейс с единственным методом (**функциональный интерфейс**), который будет вызываться на стороне принимающего команд метода.

Более того, можно взять один из существующих интерфейсов Java.

Не командуй!
Runnable

TEL-RAN
by Starta Institute

Runnable – интерфейс, имеющий единственный метод run(), который ничего не возвращает.

```
package java.lang;

@FunctionalInterface
public interface Runnable {
    public abstract void run();
}
```

Аннотация *@FunctionalInterface* обозначает, что этот интерфейс – функциональный.

Чаще всего этот метод применяется для создания потока (*Thread*) в многопоточном приложении, но для наших целей это не важно, т.к. нам просто нужен интерфейс с одним методом (может быть любой другой подходящий интерфейс).

Демонстрация класса Timer

Sorting_05.11.2023.zip

Не командуй!

Объекты Runnable

TEL-RAN
by Starta Institute

Для создания объектов типа *Runnable* нужно создать классы, имплементирующие этот интерфейс. Наиболее простым способом будет создать анонимные классы.

Демонстрация метода compareSortsWithAnonimClasses

Sorting_05.11.2023.zip

Напомню, что анонимный класс создаётся на месте вызова оператора *new* с интерфейсом или абстрактным классом. В нём необходимо написать реализацию абстрактных членов. Здесь же будет создан один единственный экземпляр этого класса. Его-то мы и будем передавать в качестве аргумента метода.

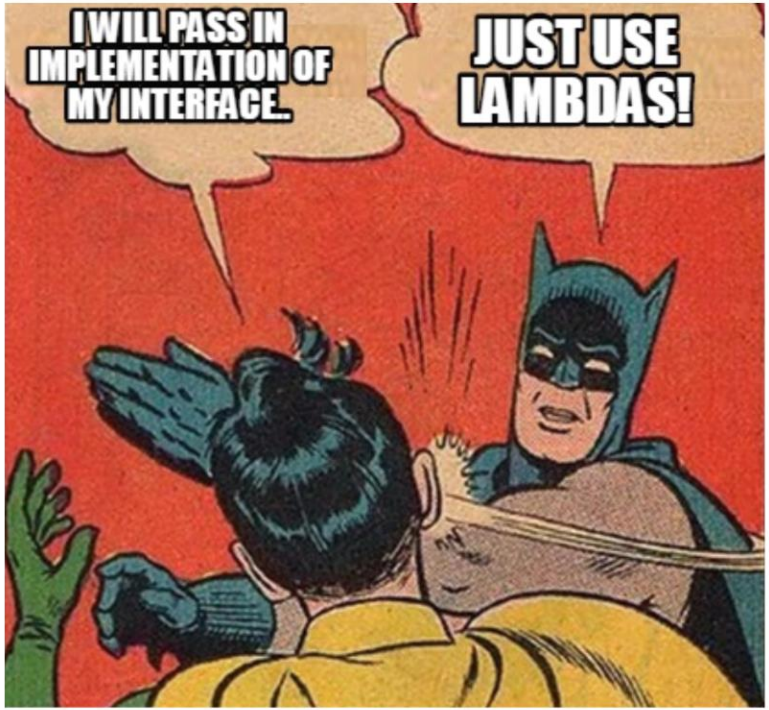
Не командуй!

Лямбда-выражения

Даже создание анонимных классов требует написания большого количества кода. Поэтому в Java 8 появился способ создавать анонимный класс (экземпляр функционального интерфейса) с помощью короткой записи, которую назвали **лямбда-выражением**.

По сути лямбда-выражение реализует тот самый единственный метод, который содержится в функциональном интерфейсе. Структура лямбда-выражения:

(аргументы метода) -> { тело }



Не командуй!

Лямбда-выражения

TEL-RAN
by Starta Institute

Входные параметры. Как и у метода, они могут отсутствовать, тогда ставим (). Допускается не указывать типы входных параметров, если они понятны из контекста. Допускается не указывать (), если аргумент только один

```
(int a, int b) -> {  return a + b; }
() -> System.out.println("Hello World");
(String s) -> { System.out.println(s); }
() -> 42
() -> { return 3.1415 };
```

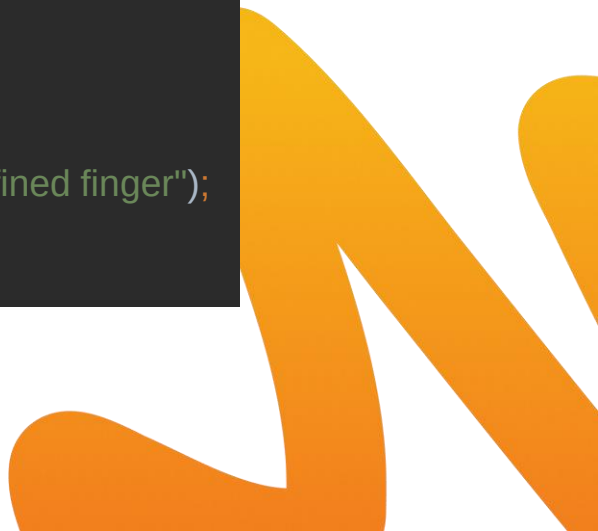
Тело метода. Если метод возвращает что-то, то указывается return. Если в теле только одна инструкция, то можно опустить return и {}.

Не командуй!

Где мы встречали лямбды

TEL-RAN
by Starta Institute

```
int num = 1;
String fingerName = switch (num) {
    case 1 -> "Big";
    case 2 -> "Index";
    case 3 -> "Middle";
    case 4 -> "Ring";
    case 5 -> "Little";
    default -> {
        System.out.println("Мутанты атакуют!");
        throw new NoSuchElementException("Undefined finger");
    }
};
```



Не командуй!

Где мы встречали лямбды

TEL-RAN
by Starta Institute

```
List<Integer> list = List.of(0, 1, 9, 2, 8, 3, 7, 4, 7);
list.forEach(e -> System.out.println(e)); // лямбда-выражение для вывода каждого элемента в консоль
list.forEach(System.out::println); // аналогичная запись с помощью ссылки на метод (компактное лямбда-выражение)

Iterator<Integer> iterator = list.iterator();
iterator.forEachRemaining(e -> System.out.println(e));
iterator.forEachRemaining(System.out::println);
```

Не командуй!

Где мы встречали лямбды

TEL-RAN
by Starta Institute

```
Map<String, Integer> nameToAge = Map.of(
        "Peter", 21,
        "Jimmy", 14,
        "Kate", 16,
        "Nina", 20
);
nameToAge.entrySet()
    .forEach(pair -> System.out.printf("I'm %s, I'm %d years old.%n", pair.getKey(), pair.getValue()));

nameToAge.forEach((k, v) -> System.out.printf("I'm %s, I'm %d years old.%n", k, v));
```

Не командуй!

Где мы встречали лямбды

TEL-RAN
by Starta Institute

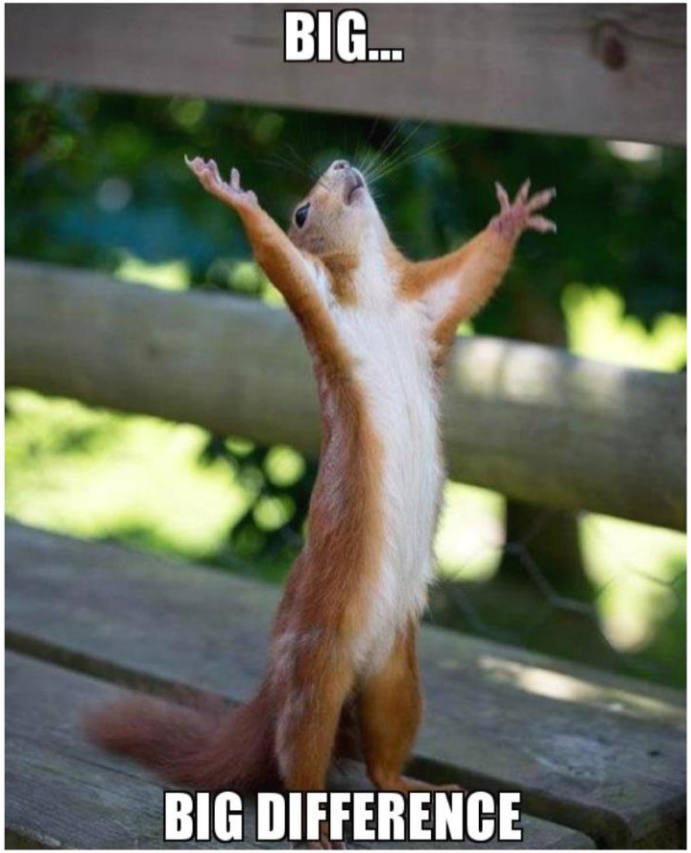
```
@Test
public void whenDerivedExceptionThrown_thenAssertionSucceeds() {
    Exception exception = assertThrows(RuntimeException.class, () -> Integer.parseInt("1a"));
    String expectedMessage = "For input string";
    String actualMessage = exception.getMessage();
    assertTrue(actualMessage.contains(expectedMessage));
}
```

Не командуй!

Отличие лямбда-выражений от анонимных классов

TEL-RAN
by Starta Institute

Главное отличие состоит в использовании ключевого слова *this*. Для анонимных классов ключевое слово *this* обозначает объект анонимного класса, в то время как в лямбда-выражении *this* обозначает объект класса, в котором лямбда-выражении используется.



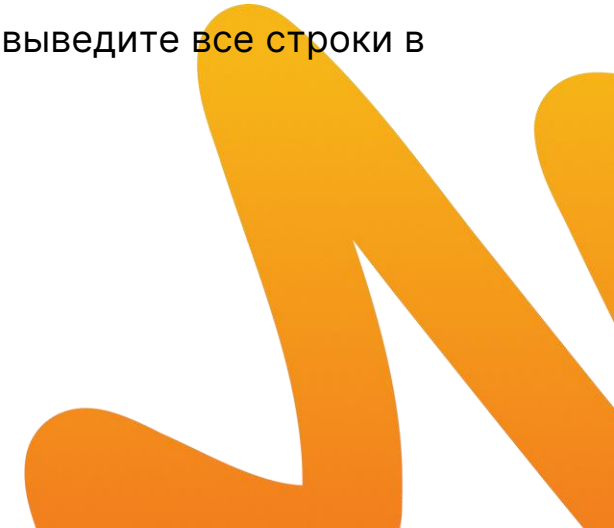
Задание

TEL-RAN
by Starta Institute

1 Напишите метод, который принимает список строк и возвращает список из тех, которые начинаются с цифры.

2 Напишите метод, который принимает мапу «Номер квартиры – Список возрастов жильцов». Метод возвращает мапу «Номер квартиры – Средний возраст жильцов».

3 Создайте 3 лямбда-выражения, которые выводят строку в консоль разными способами. Передайте все лямбда-выражения в метод в виде коллекции и выведите все строки в консоль.



Проблема

TEL-RAN
by Starta Institute

Вы, наверно, заметили, что метод *run* не возвращает результата (*void*).

Как быть, если нужно, чтобы переданное лямбда-выражение отдавало нам результат своего выполнения?

Зовём и получаем
Callable

TEL-RAN
by Starta Institute

Callable – функциональный параметризированный интерфейс, содержащий метод call(), который умеет возвращать значение. Является аналогом Runnable и тоже применяется в основном в многопоточности.

```
package java.util.concurrent;

@FunctionalInterface
public interface Callable<V> {
    V call() throws Exception;
}
```

Демонстрация MainCallable

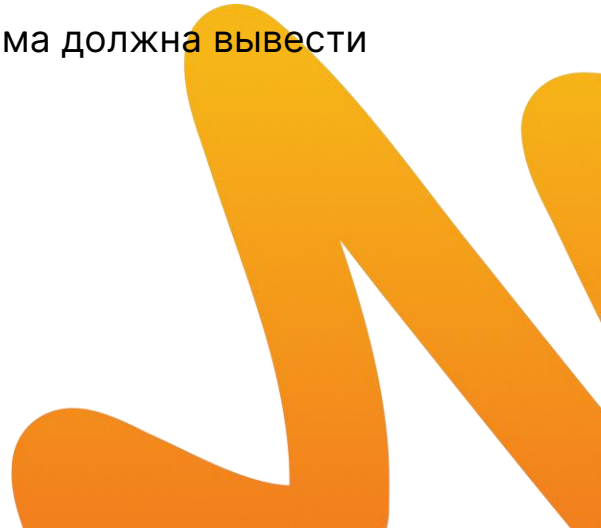
LambdaExample.zip

Задание

TEL-RAN
by Starta Institute

Напишите методы для вычисления суммы, разности, произведения и частного двух чисел. Создайте метод calculate, который будет возвращать результат вычисления, принимая в качестве параметра экземпляр Callable. Перед вычислением метод должен выводить сообщение «Выполняю вычисление», а после вычисления – «Вычисление выполнено».

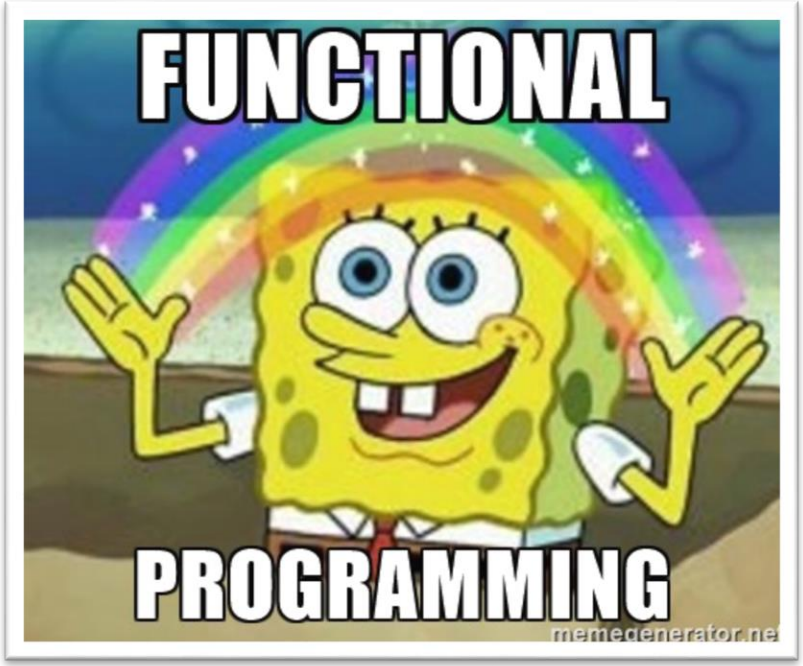
Организуйте ввод пользователя: два числа и оператор. Программа должна вывести результат в консоль и в файл.



Не командуй!

Функциональное программирование

Лямбда выражения привнесли в объектно-ориентированный язык программирования Java возможность писать код в **функциональном стиле** (см. Функциональное программирование).



Не командуй!

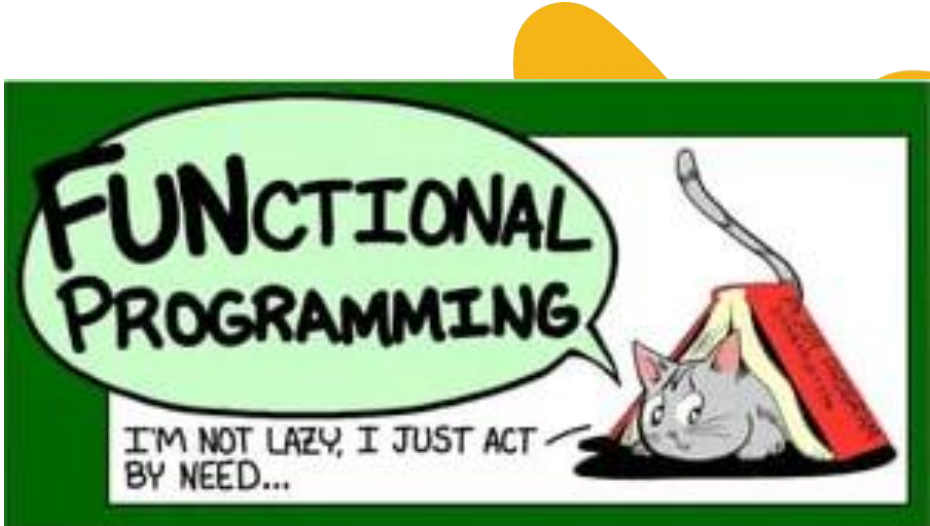
Отложенное выполнение

TEL-RAN
by Starta Institute

Одним из ключевых моментов в функциональном программировании и использовании лямбд является **отложенное выполнение (deferred execution)**. То есть мы определяем в одном месте программы лямбда-выражение и затем можем его вызывать при необходимости неопределенное количество раз в различных частях программы.

Требуется для:

- выполнения кода отдельном потоке
- выполнения одного и того же кода несколько раз
- выполнения кода в результате какого-то события
- выполнения кода только в том случае, когда он действительно необходим и если он необходим

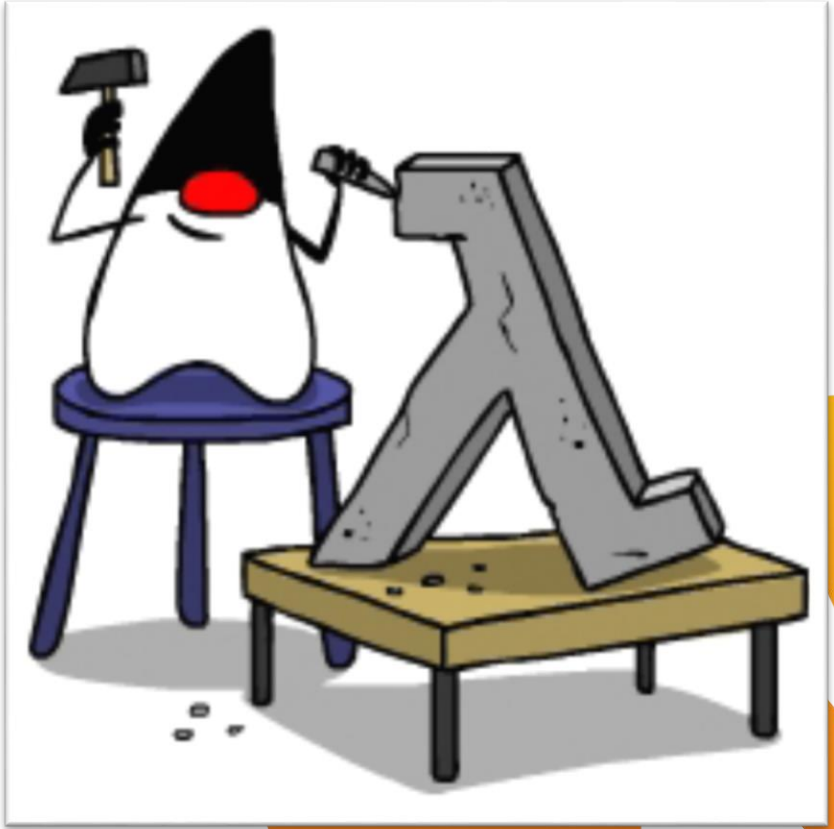


Проблема

TEL-RAN
by Starta Institute

Мы знаем теперь, что *Callable* возвращает значение, а *Runnable* нет. При этом методы обоих интерфейсов не принимают никаких значений.

А есть ли функциональные интерфейсы, которые их принимают? Какие вообще существуют функциональные интерфейсы?



И как это функционирует?

Функциональные интерфейсы

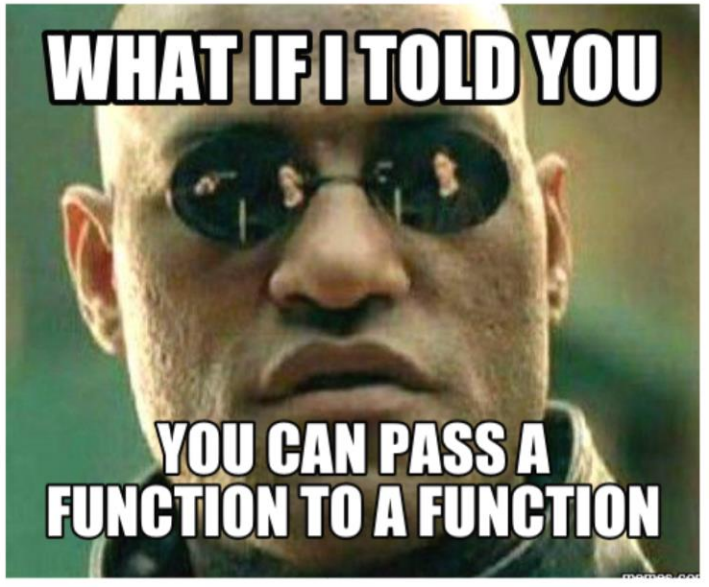
TEL-RAN
by Starta Institute

Функциональные интерфейсы (Functional Interface) – это интерфейсы только с одним абстрактным методом, объявленным в нем. Помечаются аннотацией *@FunctionalInterface*. Функциональные интерфейсы могут содержать сколько угодно методов с реализацией по умолчанию (помеченные ключевым словом *default*)

Функциональные интерфейсы – это типы данных, которые позволяют хранить лямбда-выражения.

```
Runnable r = () -> System.out.println("hello world");
```

В зависимости от того, что принимает и возвращает лямбда выражение нужно подбирать подходящий интерфейс.



И как это функционирует?

Пакет java.util.function

TEL-RAN by Starta Institute

Интерфейс	Описание
Consumer<T>	Данный интерфейс нужен для выполнения каких-то действий над объектом, при этом не нуждаясь в возвращении каких-то значений. Содержит функцию **accept**, которая принимает объект типа **T** и выполняет над ним требуемые действия.
Function<T,R>	Данный интерфейс нужен для перехода от объекта типа **T** к объекту типа **R**. Содержит функцию **apply**, которая принимает объект типа **T** и возвращает объект типа **R**.
Predicate<T>	Проверяет соблюдение некоторого условия. В интерфейсе содержится функция **test**, которая принимает аргумент типа **T**, и возвращает значение **true**, если аргумент удовлетворяет предикату, и **false** в противном случае.
Supplier<T>	Данный интерфейс содержит функцию **get**, которая ничего не принимает, но при этом возвращает значение типа **T**. Такой интерфейс применяется, например, когда нужно лямбда-выражение без аргументов.
UnaryOperator<T>	Данный интерфейс представляет собой какую-то унарную операцию. Наследуется от интерфейса **Function**.
BinaryOperator<T>	Данный интерфейс представляет собой какую-то бинарную операцию. Содержит функцию **apply** (наследуется от интерфейса **BiFunction**), которая принимает два аргумента типа **T**, выполняет над ними бинарную операцию и возвращает ее результат также в виде объекта типа **T**.
BiConsumer<T,U>	То же что и **Consumer**, только принимает два значения вместо одного.
BiFunction<T,U,R>	То же что и **Function**, только принимает два значения вместо одного.
BiPredicate<T,U>	То же что и **Predicate**, только принимает два значения вместо одного.

И как это функционирует?

Пример использования функциональных интерфейсов

TEL-RAN
by Starta Institute

FunctionalInterfaceExample.zip

Задание

TEL-RAN
by Starta Institute

1 Положите лямбда-выражения, указанные ниже, в переменные

```
s -> System.out.println(s);
x -> x*x;
s -> s == null || s.isBlank();
(age, name) -> System.out.printf("%s is %d years old", name, age);
(a, b) -> a + b;
(s1, s2) -> s1 != null && s2 != null && s1.length() > s2.length();
```

Решение

TEL-RAN
by Starta Institute

1 Положите лямбда-выражения, указанные ниже, в переменные

```
Consumer<String> print = s -> System.out.println(s);
    Function<Integer, Integer> square = x -> x*x;
    Predicate<String> isNullOrBlank = s -> s == null || s.isBlank();
    BiConsumer<Integer, String> printAsPhrase = (age, name) ->
        System.out.printf("%s is %d years old", name, age);
    BiFunction<Integer, Integer, Integer> add = (a, b) -> a + b;
    BiPredicate<String, String> isFirstLonger = (s1, s2) ->
        s1 != null && s2 != null && s1.length() > s2.length();
}
```

Задание

TEL-RAN
by Starta Institute

2 Создайте методы, реализующие вычисление математических функций:

$y = x^2; y = x^3; y = \sqrt{|x|}; y = \sin x;$

Напишите метод, который принимает математическую функцию и возвращает список вычисленных значений в диапазоне значений от $-\frac{\pi}{2}$ до $\frac{\pi}{2}$.

И как это функционирует?

Свои функциональные интерфейсы

Вы можете создавать свои функциональные интерфейсы. Достаточно объявить всего одни метод в таком интерфейсе. Рекомендуется помечать такой интерфейс аннотацией @FunctionalInterface.

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

И как это функционирует?

Особенность функциональных интерфейсов

TEL-RAN
by Starta Institute

Функциональный интерфейс может быть обобщенным, однако в лямбда-выражении использование обобщений не допускается. В этом случае нам надо типизировать объект интерфейса определенным типом, который потом будет применяться в лямбда-выражении.

```
public class LambdaNotGeneric {
    public static void main(String[] args) {
        Operationable<Integer> operation1 = (x, y)-> x + y;
        Operationable<String> operation2 = (x, y) -> x + y;
        System.out.println(operation1.calculate(20, 10)); //30
        System.out.println(operation2.calculate("20", "10")); //2010
    }
}
interface Operationable<T>{
    T calculate(T x, T y);
}
```

Задание

TEL-RAN
by Starta Institute

Создайте собственный функциональный интерфейс, внутри которого определите метод execute, принимающий три аргумента типа T и возвращающий результат типа R.

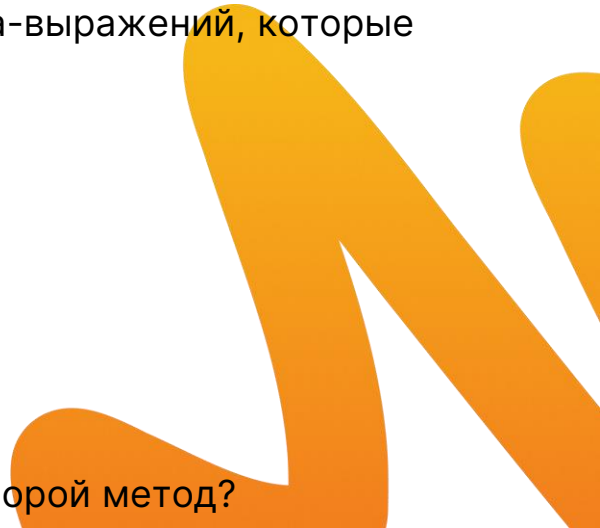
В Main создайте метод, checkAndGet, принимающий экземпляр данного интерфейса и три строки. Если хотя бы одна из переданных строк null, метод возвращает null. В противном случае вызывает метод execute.

В методе main создайте три строки. Создайте несколько лямбда-выражений, которые принимают три строки и возвращают:

- суммарную длину строк
- среднюю длину строк
- длину наименьшей строки
- длину строки, в которой больше всего цифр.

Вызовите метод checkAndGet для всех лямбда-выражений.

Что произойдёт, если добавить в функциональный интерфейс второй метод?



TEL-RAN
by Starta Institute

3 Домашнее задание

Домашнее задание

TEL-RAN
by Starta Institute

1 Создайте список чисел. С помощью метода forEach выведите каждое число в консоль по модулю.

2 Создайте класс Item с полями названия товара и стоимости товара. Создайте мапу, хранящую товар и соответствующее ему число единиц товара. С помощью метода forEach получите все товары, у которых осталось меньше 3 штук.

3 Создайте функциональный интерфейс для генерации случайных чисел в заданном диапазоне. Используйте лямбда-выражение для генерации числа.

4 Создайте предикат для удаления из списка тех строк, что начинаются с определённой буквы. Создайте список строк и удалите элементы с помощью метода removeIf, передав в него предикат.

ЗАКЛЮЧЕНИЕ

TEL-RAN
by Starta Institute