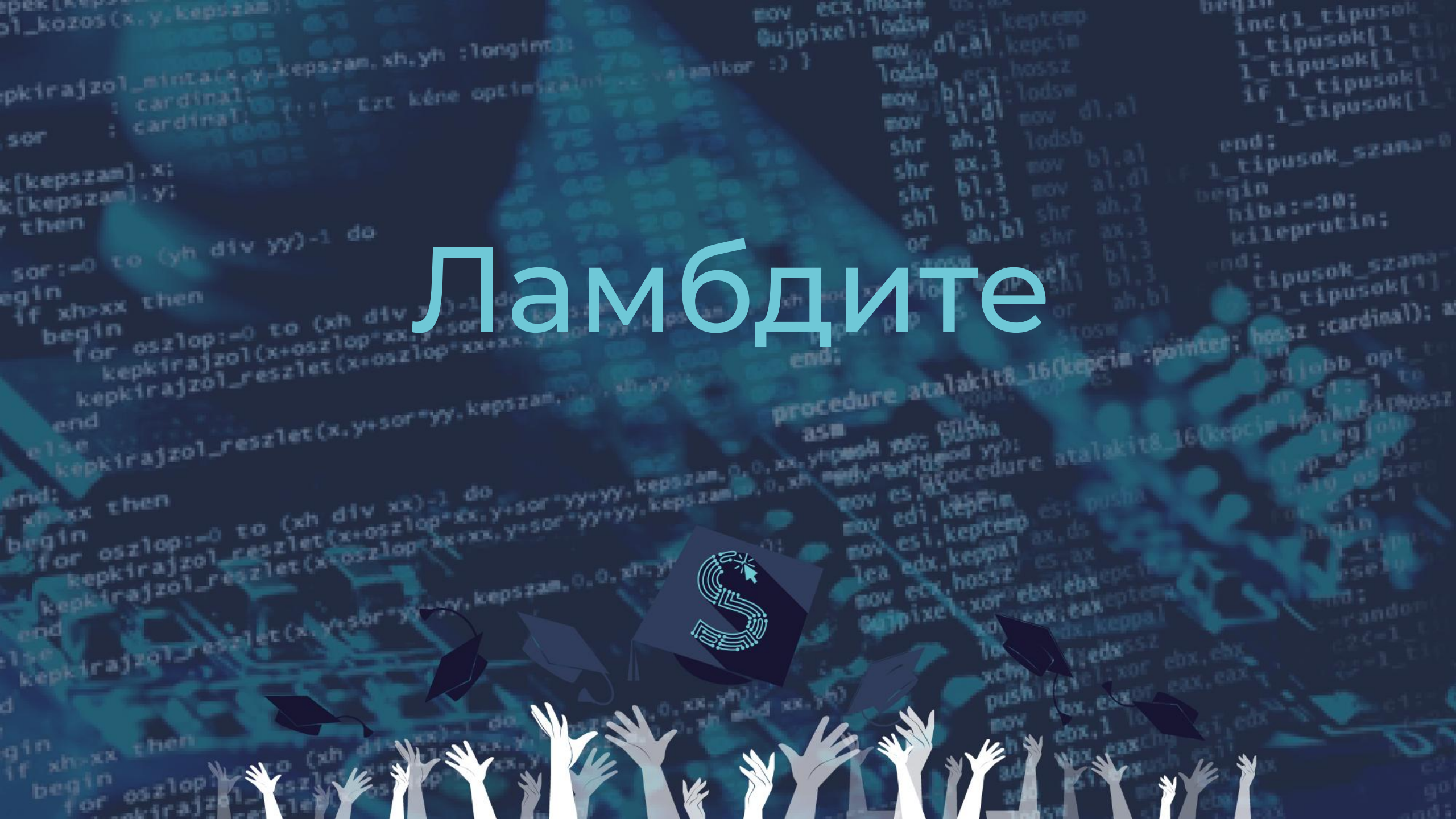
ЛЕКЦИЯ I

JAVA OOP: Ламбди и Потоци

Теодор Костадинов

Съдържание

Ламбдите



Функционални интерфейси

- Функционалните интерфейси са интерфейси с точно един абстрактен метод
- Понякога се наричат още SAM интерфейси (Single Abstract Method))

SAM:

```
interface Runnable {
    public void run();
}
```

- Сортирайте следният списък по последна буква на имената:

```
List<String> names = Arrays.asList("Peter", "Anna", "Mike", "Xenia");
```

Решение


```
List<String> names = Arrays.asList("Peter", "Anna", "Mike", "Xenia");

Collections.sort(names, new Comparator<String>() {
    @Override
    public int compare(String a, String b) {
        return a.compareTo(b);
    }
});

// Налага се да създадем отделен клас, имплементиращ интерфейс,
// да създадем обект от него и да го подадем като аргумент.
// Има ли начин да подадем само дефиниция на функция като
аргумент?
```

Ами сега?

Задача за упражнение

Ламбди

- Функция, която няма име и не е метод на клас, се нарича ламбда функция или ламбда израз.
- Ламбдите заместват методите в анонимни класове
- Ламбдите правят кода по-четим
- Ламбдите позволяват навръзването на няколко една след друга
- Спестяват код

Без lambda:

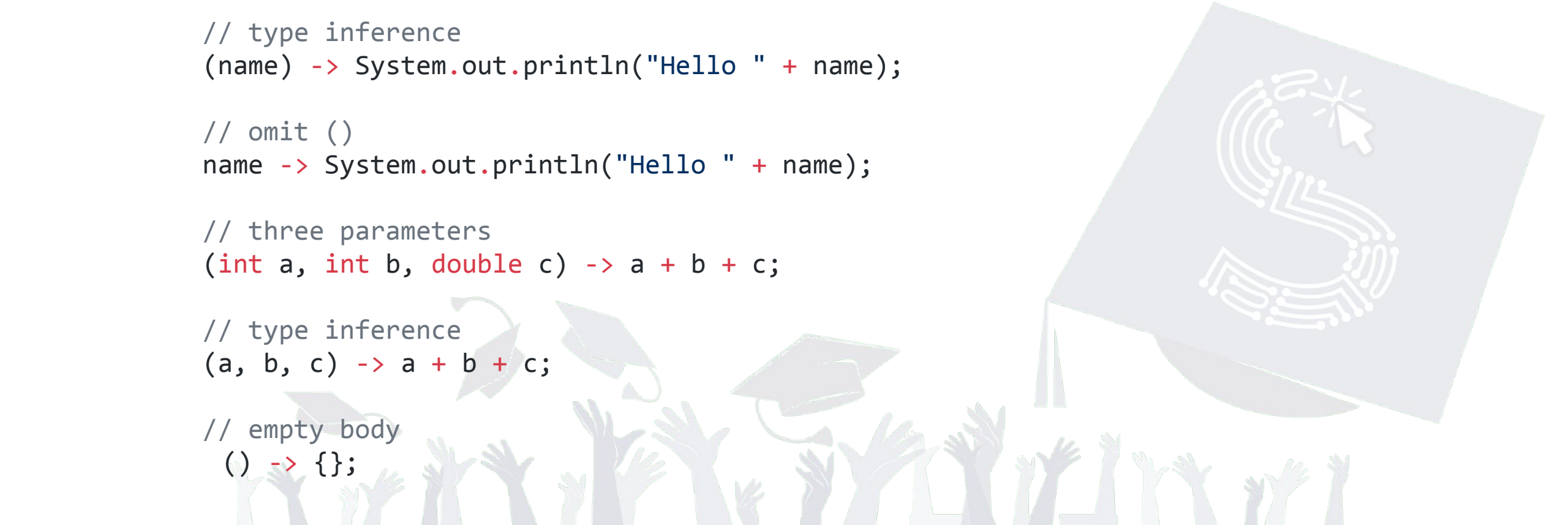
```
Thread th;
th = new Thread(new Runnable() {
    public void run() {
        doSomething();
    }
});
th.start();
```

С lambda:

```
Thread th;
th = new Thread(() -> doSomething());
th.start();
```

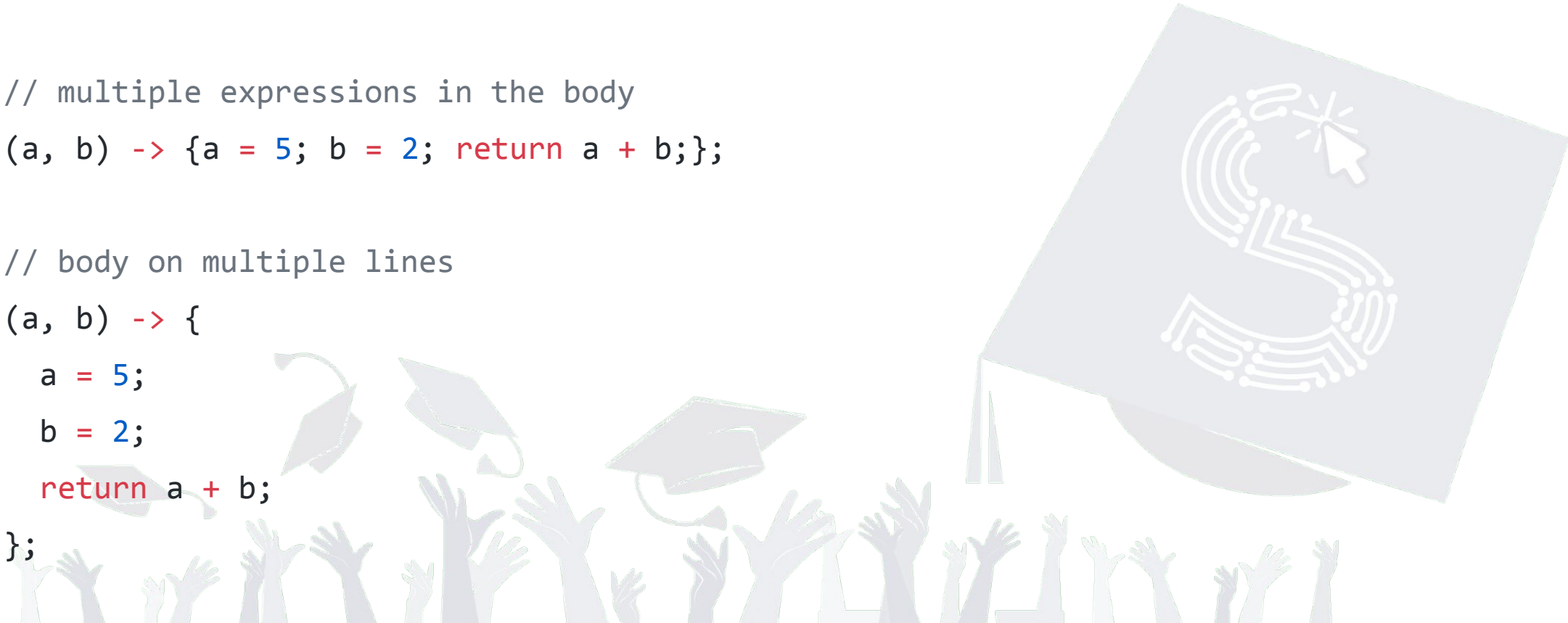
Примери


```
// one parameter
(String name) -> System.out.println("Hello " + name);

// type inference
(name) -> System.out.println("Hello " + name);

// omit ()
name -> System.out.println("Hello " + name);

// three parameters
(int a, int b, double c) -> a + b + c;

// type inference
(a, b, c) -> a + b + c;

// empty body
 () -> {};
```



Примери

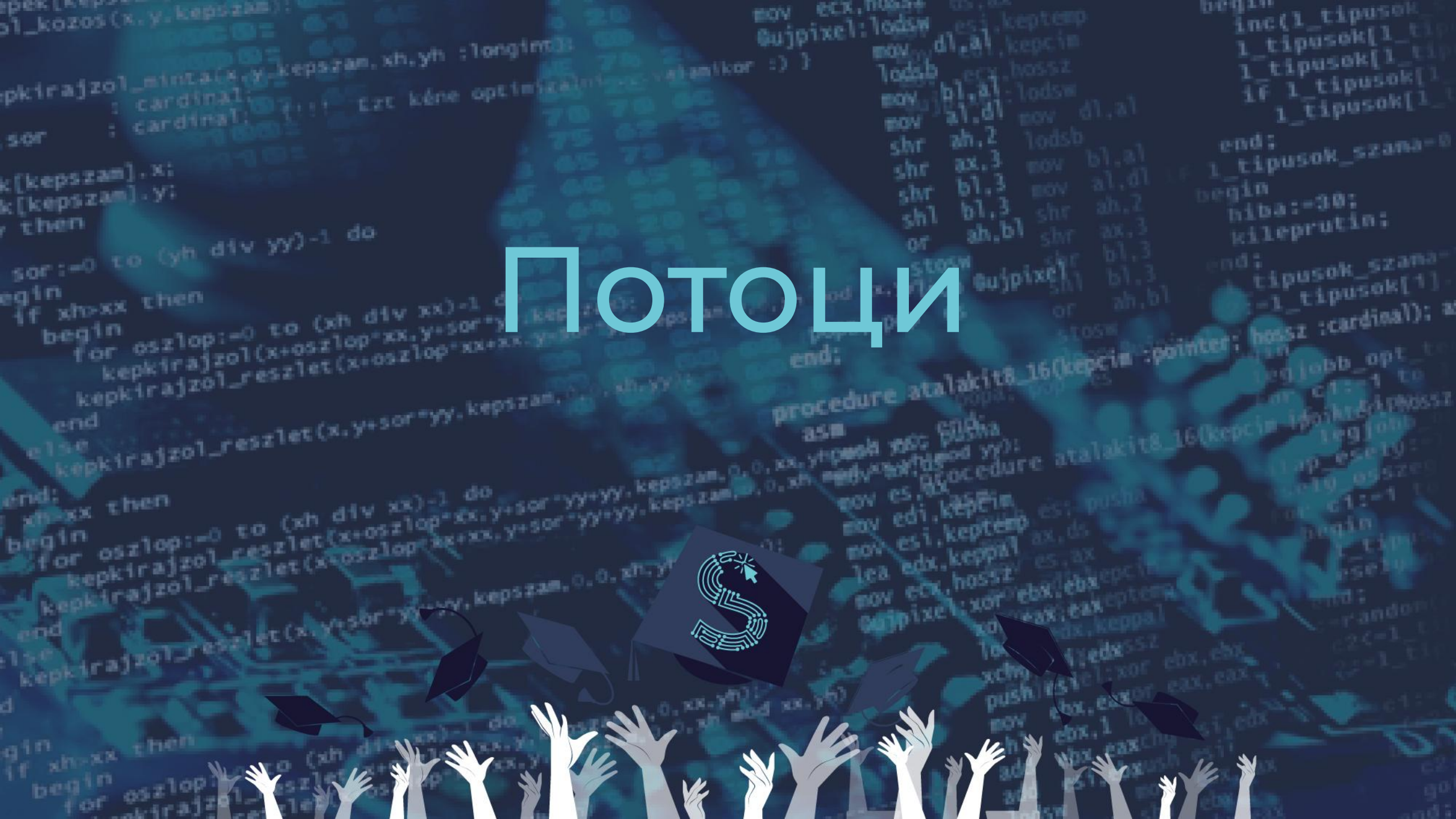

```
// omit {} and the return as there is single expression in the body
(a, b) -> a + b;

// multiple expressions in the body
(a, b) -> {a = 5; b = 2; return a + b;};

// body on multiple lines
(a, b) -> {
  a = 5;
  b = 2;
  return a + b;
};
```



Препоръки

- Дръжте ламбдите си кратки - могат да бъдат блокове код, но не трябва
- Те са просто връзката между кода
- 1 ред ламбда е повече от достатъчен
- Ламбдите са във формат: параметри -> тяло
- Списъкът с параметри се огражда с кръгли скоби, а параметрите се разделят със запетаи. Може да е празен ()
- Типът на параметрите може да бъде
  - експлицитно зададен
  - inferred: с var или въобще пропуснат
- Когато има един-единствен параметър и неговият тип може да бъде пропуснат, може да се пропуснат и скобите
- Тялото на ламбда израз се огражда с фигурни скоби {} и може да съдържа нула, един или повече изрази, разделени с ;
  - Когато има единствен израз в тялото, return клаузата може да се изпусне
  - Когато имаме повече от един израз, трябва експлицитно да предоставим return клауза

Потоци



Потоци

SOFTWARE ACADEMY
Educational Center Net It

- Концепцията за вход-изход в Java се основава на потоци (streams)
- Потокът е абстракция за безкраен поток от данни
- Може да се четат данни от поток или да се пишат данни в поток
- В Java потоците може да се основават на байтове или на символи

Stream.API

- Ламбда изразите, заедно с функционалните интерфейси, разширяват възможностите на Java с елементи на функционалното програмиране.
- Те позволяват предаването на поведение (функции) като параметри на библиотеки, оптимизирани за бързодействие при обработката на данни.
- По този начин, един app developer може да се фокусира върху бизнес логиката на приложението си, оставяйки аспекти като бързодействието на авторите на въпросните библиотеки. Една такава основна библиотека е `java.util.stream`.
- Интерфейсите и класовете от пакета `java.util.stream`, които съвкупно наричаме Java Stream API, са предназначени за ефективната последователна или паралелна обработка на крайни или безкрайни потоци от данни.
- Алгоритмите, работещи с данни във вид на потоци, се реализират като последователност (pipeline) от операции върху елементите на потока.

Потоци

- Потокът не е структура от данни. Той приема данни от List, Array, Map и ги препредава
- Той е метод за предаване на данни от източник, през серия от операции, до крайна операция, която създава резултата
- Потоците не променят оригиналната структура на данните, те само предоставят резултата

```
taxiTrips
                .map { it.driver to it.passengers }
                .filter { p -> p.first == driver }
                .flatMap { p -> p.second }
                .groupBy { p -> p.name }
                .filter { (_, k) -> k.size > 1 }
                .flatMap { (_, v) -> v }
                .toArray();
```

Два вида операции могат да се извършват върху поток

- Intermediate Operations
- Terminal Operations

Създаване на поток


```
Stream.empty() // създава празен поток (без елементи)

Stream.of(T... values) // създава поток от елемееенти от тип T

Collection.stream() // връща поток от елементите на дадената колекция

// връща паралелен поток от елементите на дадената колекция
Collection.parallelStream()

Arrays.stream((T[] array)) // връща поток от елементите на масив
```

Междинни операции върху потоци

- Има три основни ТИПА междинни операции
  - map -> променат елементите на входния поток - Те map-ват, (трансформират) всеки елемент на входния поток към нов елемент - Map е междинна операция, приема функция (T -> V) и връща поток със същия брой елементи, но от новия тип (V)
  - filter -> филтрира елементите от потока
  - sort -> сортира елементите
- Има терминиращи операции, които спират потока

```
List<Person> result = people.stream()
        .filter(p -> p.getFirstName().equals("Nikolay"))
        .filter(p -> p.getAge() < 25)
        .sorted(Comparator.comparing(Person::getLastName))
        .collect(Collectors.toList());
```

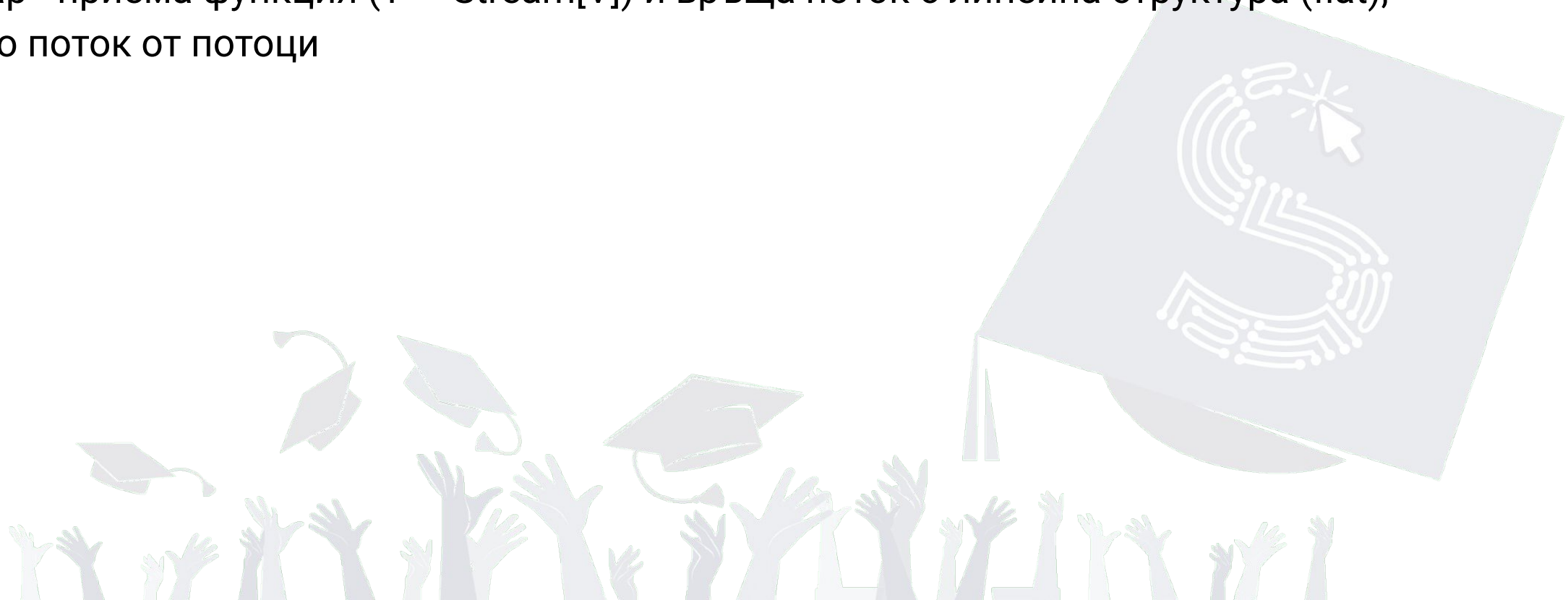
Филтриращи операции

- Filter - приема фунцкия (T -> boolean), т.е. Предикат и връща поток само с елементите, за които предикатът е true.
- Limit - приема цяло число N и връща краен поток само с първите N елемента на входния поток. Особено често се използва, за да се обработи крайна част от безкраен поток
- Distinct - операция без аргументи, връща елементите на входния поток като премахва повтарящите се. Критерият за еднаквост е equals()
- Skip - приема цяло число N и връща краен поток, игнорирайки (пропускайки) първите N елемента на входния поток
- DropWhile - пропуска първите елементи на потока, докато предикатът връща true
- TakeWhile - допуска само първите елементи на потока, докато предикатът връща true

Мапващи операции

SOFTWARE ACADEMY
Educational Center Net It

- Map - проста операция за мапване
- FlatMap - приема функция (T -> Stream[V]) и връща поток с линейна структура (flat), вместо поток от потоци



Сортиращи операции

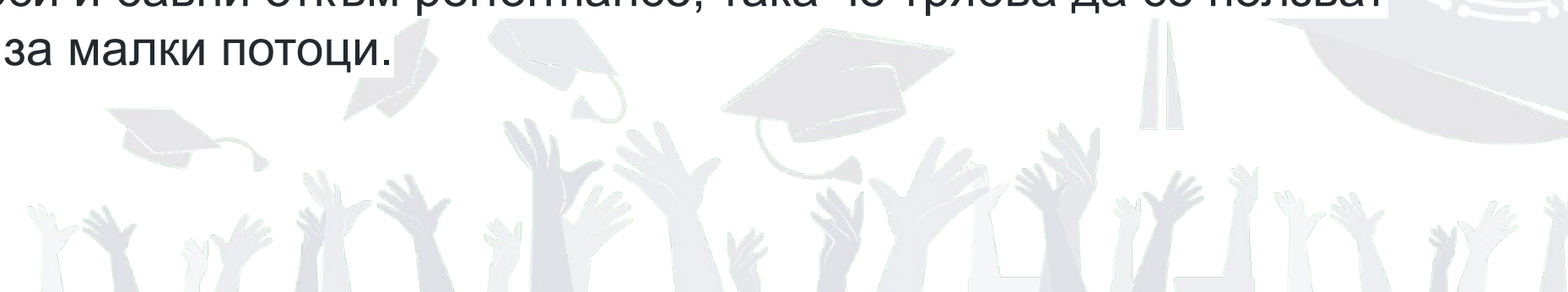
- Според пре-дефинираният `equals` метод на обектите

```
employees.sorted();
```

- Според експлицитно зададено правило

```
employees.sorted((e1, e2) -> e1.getSalary() < e2.getSalary());
```

Обърнете внимание, че тези операци не могат да приключат докато не се обработят всички елементи на потока. Те са скъпи откъм ресурси и бавни откъм performance, така че трябва да се ползват само за малки потоци.



Терминиращи операции

- Терминална операция, приема функция (T -> void) и не връща резултат (т.е. няма функционална природа)
- Foreach е подобна функция
- Reduce - Терминална операция, приема функция ((T, T) -> T) и връща единичен резултат
  - Min / Max
- Collect - Възможно е reduce операция да върне резултат, който не е единичен обект, а колекция - collect предоставя възможност за акумулиране на резултата в колекция
- Count - връща броя
-

Терминиращи операции - преждевременни

- findFirst - връща първия елемент на потока
- findAny - връща произволен елемент от потока
- allMatch(T -> boolean) – дали всички елементи отговарят на дадено условие
- anyMatch(T -> boolean) – дали някой елемент отговаря на дадено условие
- noneMatch(T -> boolean) - дали никой от елементите не отговаря на дадено условие

Почивка

до 20:25

Таксиметрова компания

Вие сте статистически анализатор на таксиметрова компания. Дадени са ви данни за шофьорите, пътниците и пътуванията.

```
public Set<Driver> allDrivers;
public Set<Passenger> allPassengers;
public List<Trip> trips;
```

Шофьорите и пътниците имат само имена. Едно пътуване има шофьор, сет от пътници, продължителност в минути, дистанция в километри, процент отстъпка от цената (0 - 1) или null, цена.

Ами сега?

Задача за упражнение

Таксиметрова компания 1

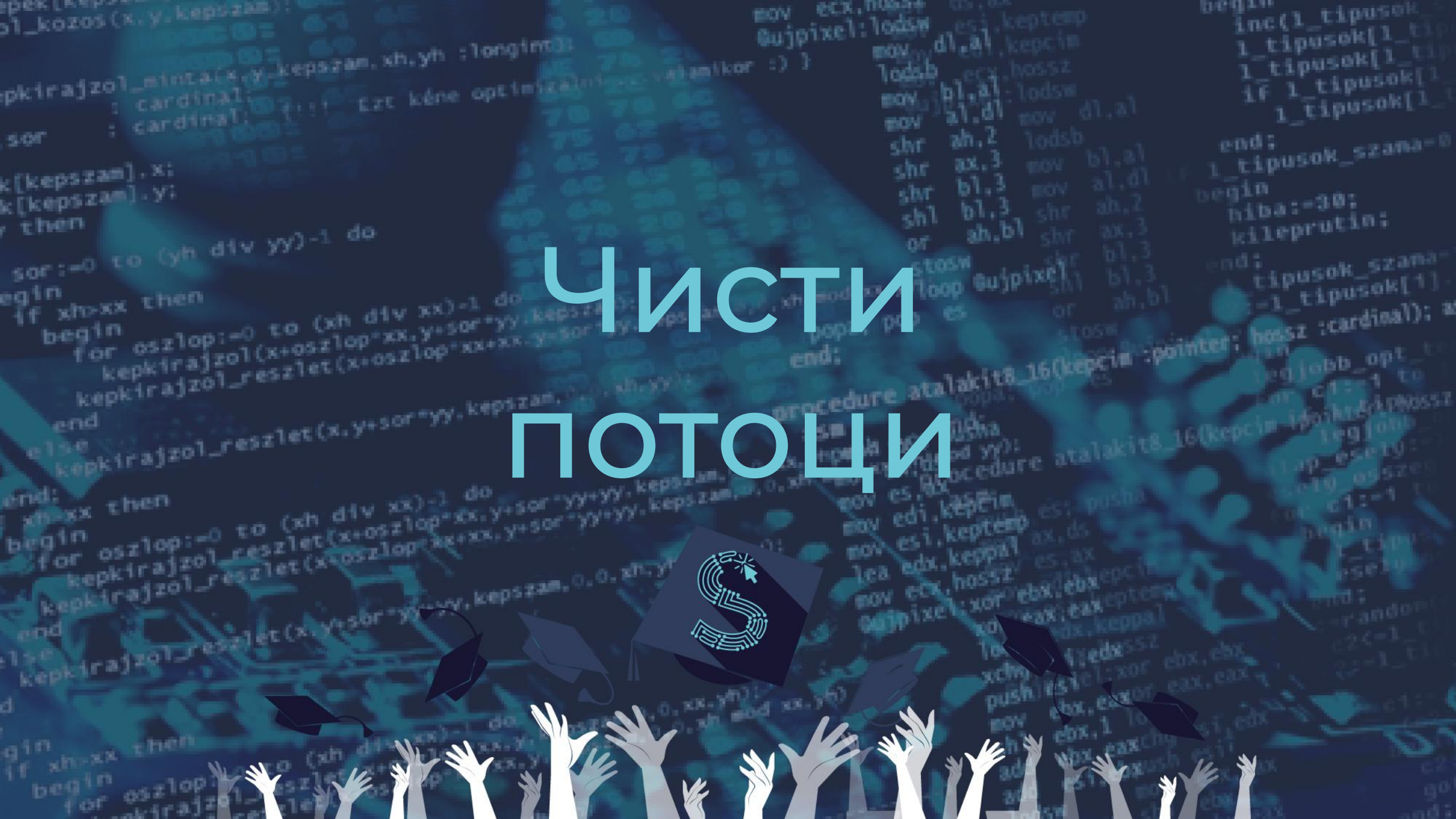
SOFTWARE ACADEMY
Educational Center Net It

Изведете списък на шофьорите-измамници. Това са шофьори, които нямат извършени пътувания.

Ами сега?

Задача за упражнение

Чисти потоци



Странични ефекти

SOFTWARE ACADEMY
Educational Center Net It

- Страничен ефект е всяко действие във функция, което променя външно за нея състояние:
  - промяна на данни (външни за функцията)
  - IO операция
  - хвърляне на изключение
  - извикване на друга функция, която предизвиква страничен ефект
- Всеки метод в Java, който е от тип void, извършва страничен ефект.

Чисти функции

- Чистите функции не извършват странични ефекти
- Винаги когато извикаме функцията с дадени параметри, е гарантиран един и същ резултат
- Резултатът им може да се запази и преизползва
- Извикването на функцията може да бъде заменено с резултата (referential transparency)
- Чистите функции са лесно композируеми
- Чистите функции са лесни за анализ на проблеми
- Позволяват изчислението да се паралелизира

ВЪПРОСИ?

Резюме

- Ламбдите спестяват код и го правят по-четим
- Потоците спестяват код и го правят по-четим

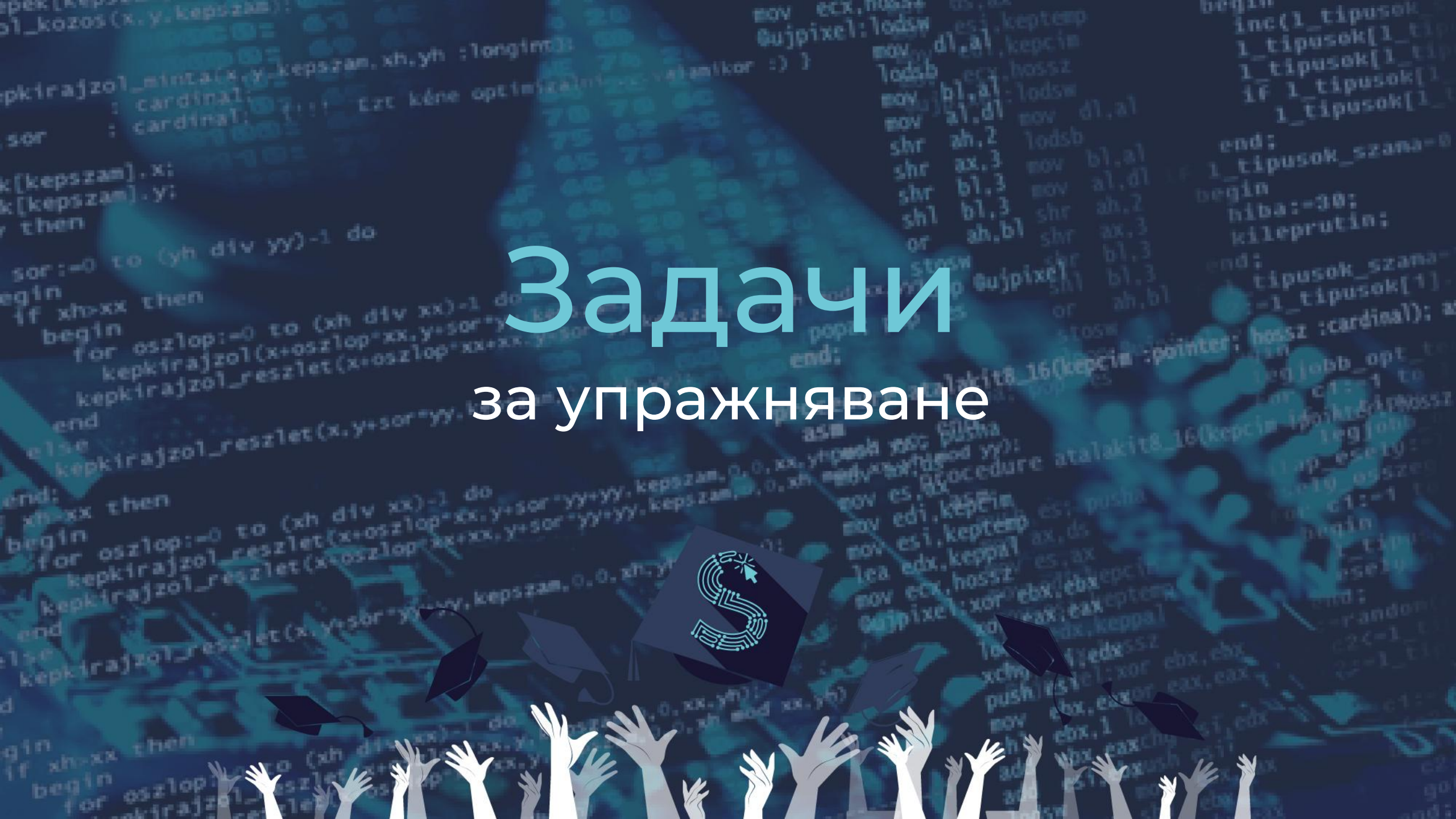
Ресурси

SOFTWARE ACADEMY
Educational Center Net It

- Stream Docs
- GitHub Repo with Demos
- University Java Course Resources

Задачи

за упражняване



Задача 2

Продължете задачата с таксиметровата компания.

Намерете всички пътници, които са извършили поне Н пътувания, където Н се въвежда.

Това домашно влиза в крайната ви оценка!

Домашно

Качвайте домашното си в ГитХъб и слагайте линка тук:

https://forms.gle/AcvCptCbSDizr2Ay6

Задача 3

Продължете задачата с таксиметровата компания.

Намерете всички пътници, които са се возили при шофьор Н повече от веднъж. Н се подава.

Задача 4

Продължете задачата с таксиметровата компания.

Намерете всички пътници, които са ползвали отстъпка при повечето от пътуванията си.

Задача 5

Продължете задачата с таксиметровата компания.

Намерете каква е най-честата продължителност на пътуванията. Групирайте пътуванията по десетици:

- Пътуване 2 мин е в група 0-9
- Пътуване 13 мин е в група 10-19
- Пътуване 44 мин е в група 40-49 и тн

Върнете като отговор групата.

Ако в две групи има еднакъв брой пътувания, върнете която и да било от двете.
Ако няма пътувания, върнете null.

Ами сега?

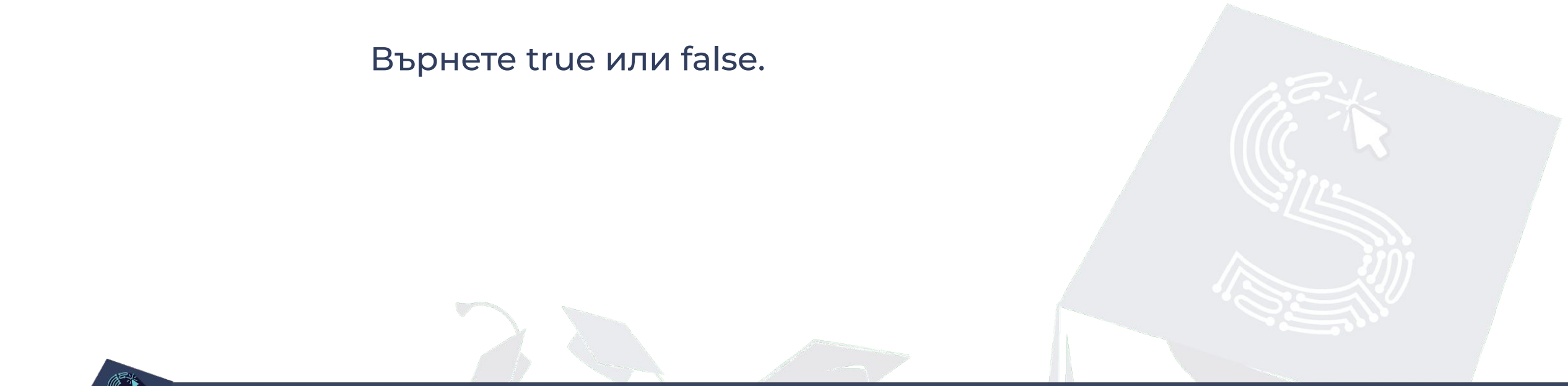
Задача за упражнение

Задача 6

Продължете задачата с таксиметровата компания.

Проверете дали 20% от шофьорите докарват 80% от приходите.

Върнете true или false.



БЛАГОДАРЯ
ЗА ВНИМАНИЕТО!