8

BALTIC TALENTS ACADEMY

JAVA LAMBDA IŠRAIŠKA

TESTAS #15 - ENUM, GENERIC

- enum
- Parametrizuotos klasės / sąsajos
- Parametrizuoti metodai
- Nežinomi tipai <?>

LAMBDA IŠRAIŠKOS

- Tarkime turime sąrašą darbuotojų ir norime atrinkti tik tuos, kurių atlyginimas didesnis negu x, kaip galima padaryti?
  - fiksuotas ciklas
  - interfeisas kaip filtras
  - lambda išraiška

Demo 1

LAMBDA IŠRAIŠKOS

- Kada lambda išraiška atitinka interfeisą:
  1. Interfeisas turi tik vieną abstraktų metodą
  2. Lambda išraiškos parametrų skaičius atitinka interfeiso metodo parametrus
  3. Lambda rezultato tipas atitinka interfeiso metodo rezultato tipą
- Tokius interfeisus, kurie atitinka lambda reikalavimus ir kurie bus naudojami lambda išraiškose galima pažymėti (anotuoti) @FunctionalInterface

LAMBDA OBJEKTAI

- Lambda išraišką galima naudoti ne tik kaip parametrą, bet ir kaip kintamojo, kurio tipas yra funkcinis interfeisas, reikšmę.
- Tokį kintamąjį galima naudoti kaip parametrą arba interfeise nurodytą metodą kviesti tiesiogiai

Demo 2

LAMBDA KINTAMŲJŲ SRITIS (SCOPE)

- Lambda išraiška gali naudoti visus kintamuosius, kurie yra prieinami išraiškos vietoje (taip ir this bei super).
- Visi panaudoti lokalūs kintamieji turi būti pažymėti final ar tokie būti pagal kontekstą, t.y. jo reikšmė po priskyrimo neturi būti keičiama

```
final int num = 1;
Converter<Integer, String> stringConverter =
        from -> String.valueOf(from + num);
stringConverter.convert(2);
```

Demo 3

LAMBDA KINTAMŲJŲ SRITIS (SCOPE)

- Bet objekto (instance) laukai gali būti naudojami ir modifikuojami be jokių apribojimų:

```
class A {
    int num = 1;
    void methodA() {
        Converter<Integer, String> stringConverter =
            from -> String.valueOf(from + ++num);
        stringConverter.convert(2);
    }
```

Demo 4

LAMBDA KINTAMŲJŲ SRITIS (SCOPE)

- Nors funkciniame interfeise galima turėti default metodus bet juos naudoti lambda išraiškoje negalima
- Kodėl?

Demo 5

STATINIŲ METODŲ PRISKYRIMAS

- Jei kažkurioje klasėje turime apsirašę statinį (!!!) metodą, tai jį galima priskirti lambda objektui naudodami užrašą:

klasė::*metodas*

- Priskirti metodą galima tik tada jei pilnai sutampa parametrai ir rezultato tipas

```
@FunctionalInterface
interface Converter<F, T> {
    T convert(F from);
}
Converter<String, Integer> converter = Integer::valueOf;
Integer converted = converter.convert("123");
```

Demo 6

KONSTRUKTORIŲ PRISKYRIMAS

- Galime priskirti taip pat ir konstruktorių.
- Užrašymui naudojame tą patį principą tik vietoj statinio metodo pavadinimo rašome new:

klasė::**new**

- Java pati atrinks reikiamą konstruktorių pagal parametrus

Demo 7

OBJEKTO METODŲ PRISKYRIMAS

- Taip pat galime priskirti ne tik statinį metodą, bet ir objekto (instance) metodą
- Užrašymui naudojame tą patį principą tik vietoj klasės naudojam objektą:

objektas::*metodas*

Demo 8

NUORODOS

- https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html
- http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/Lambda-QuickStart/index.html
- http://winterbe.com/posts/2014/03/16/java-8-tutorial/

UŽDAVINYS

Parašykite parametrizuotą (generic) klasę, kuri realizuoja E tipo objektų saugyklą. Taip pat padarykite kad juos galima iteruoti nurodant su lambda išraiška, palyginančia du elementus, kokia tvarka bus iteruojame.

Pvz. jūsų klasė-konteineris vadinasi Container ir mes į jį dedame Employee tipo objektus. Tada maždaug toks kodas turėtų veikti:

```
Container<Employee> box = new Container<>();
box.add(new Employee(…));
box.add(new Employee(…));
…

for (Employee emp : box.order((e1, e2) -> { … })) {
    System.out.println(emp);
}
```