Jazyk Java 1

Seminář 8

Daniel Bazala

Katedra informatiky
Univerzita Palackeho v Olomouci

Lambda výrazy

Lambda výrazy

- umožňují jednoduše definovat funkcionalitu
- místo anonymních tříd, které jsou složitější
- můžeme předávat jako argument metodě
- př.

```
(int x, int y) -> x + y
() -> 42
(String s) -> { System.out.println(s); }
() -> { return 3.1415 };
```

Funkční rozhraní

- Rozhraní Runnable

```
Runnable r = () -> System.out.println("Hello!");
r.run(); // Hello!
```

- Rozhraní Supplier

```
Supplier<Integer> s = () -> 42;
System.out.println(s.get());
```

- Rozhraní Consumer

```
Consumer<Integer> c = (Integer x) -> System.out.println("I␣" + x);
c.accept(42); // I 42
```

- Rozhraní Callable podobné jako Supplier, ale může vyhodit výjimku

Funkční rozhraní

- Rozhraní Function

```
Function<Double, Double> f = r -> 2 * 3.14 * r;
Double d = f.apply(2.1);
System.out.println(d); // 13.188
```

- Rozhraní BiFunction

```
BiFunction<Double, Double, Double> f = (a, b) ->
        Math.sqrt(Math.pow(a, 2) + Math.pow(b, 2));
Double c = f.apply(3.0, 4.0);
System.out.println(c); // 5.0
```

- Rozhraní Predicate

```
Predicate<Integer> p = x -> x > 42;
System.out.println(p.test(1)); // false
```

Přehled

- Funkční rozhraní

```
Supplier        ()     -> x
Consumer        x      -> ()
Callable        ()     -> x throws ex
Runnable        ()     -> ()
Function        x      -> y
BiFunction      x,y    -> z
Predicate       x      -> boolean
UnaryOperator   x1     -> x2
BinaryOperator  x1,x2  -> x3
```

Vlastní funkční rozhraní

- Rozhraní pouze s jednou metodou

```
interface Functional {
  int function(int a, int b, int c);
}

Functional f = (x, y, z) -> x + y + z;
int r = f.function(1, 2, 3);
System.out.println(r); // 6
```

Předání jako argument metody

```
public static void applyIf(List<Integer> list,
Predicate<Integer> p, Function<Integer, Integer> f) {
  for (int i = 0; i < list.size(); i++) {
    Integer e = list.get(i);
    if (p.test(e))
      list.set(i, f.apply(e));
  }
}
```

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
System.out.println(numbers); // [1, 2, 3, 4, 5]
applyIf(numbers, x -> x % 2 == 0, x -> 2 * x);
System.out.println(numbers); // [1, 4, 3, 8, 5]
```

- Př.

```
// Starý způsob
List list = Arrays.asList(1, 2, 3, 4, 5, 6, 7);
for(Integer n: list) {
    System.out.println(n);
}

// Pomocí lambda výrazu:
List list = Arrays.asList(1, 2, 3, 4, 5, 6, 7);
list.forEach(n -> System.out.println(n));

// Syntaktický cukr v Java 8
list.forEach(System.out::println);
```

- Př. výpis emailů z kolekce lidí podle kritérií

```
public static <X, Y> void processElements(
    Iterable<X> source,
    Predicate<X> tester,
    Function <X, Y> mapper,
    Consumer<Y> block) {
    for (X p : source) {
        if (tester.test(p)) {
            Y data = mapper.apply(p);
            block.accept(data);
        }
    }
}

processElements(
    persons,
    p -> p.getGender() == Person.Sex.MALE
        && p.getAge() >= 18
        && p.getAge() <= 25,
    p -> p.getEmailAddress(),
    email -> System.out.println(email)
);
```

Přehled

- Více: https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html

Stream API

Stream API

- inovativní přístup k práci s kolekcemi
- kolekce jsou chápány pouze jako uložiště pro data
- podpora líného vyhodnocování a implicitního paralelismu (na rozdíl od běžného iterování)
- kořeny ve funkcionálním programování
- využití zejména v kombinaci s lambda výrazy

Stream API

- přehled: https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Stream.html
- docs https://docs.oracle.com/javase/tutorial/collections/streams/index.html
- vytvoření streamu

```
Stream.of(1, 2, 3) // stream hodnot
Stream.of(fooArray) // vytvoreni streamu z pole
foo.stream() // vytvoreni streamu z kolekce
stream.map // vytvoreni z jineho streamu
stream.filter
```

Stream API

- metoda filter - restrikce podle kritérií (predikátu)
- metoda map - mapování podle funkce
- metody mapToInt/mapToDouble/.. - vrací stream integerů/doublů
- metody anyMatch/allMatch/noneMatch - otestování všech prvků
- metoda distinct - vrací stream s unikátními prvky
- metody sum, average, min, max, count, ...
- metoda reduce - redukce na jednu hodnotu podle funkce

Stream API

- př.

```
Stream<Integer> s = Stream.of(1, 2, 3, 4, 5);
int r = s.filter(x -> x < 4).mapToInt(x -> 2 * x).sum();
System.out.println(r); // 12

roster
   .stream()
   .forEach(e -> System.out.println(e.getName());

double average = persons
   .stream()
   .filter(p -> p.getGender() == Person.Sex.MALE)
   .mapToInt(Person::getAge)
   .average()
   .getAsDouble();
```

Úkol

Úkol seminář 8

- http://marcus.webly3d.net/ukol8