

# Java 8: ламбда функции и Stream API

Любомир Чорбаджиев

Технологично училище “Електронни системи”  
Технически университет, София

24 март 2017 г.



# Съдържание

## 1 Ламбда функции

# Пример: филтриране на списък от хора

- Дефиниция на класа Person:

```
1 public class Person {  
2  
3     private String names;  
4  
5     private Gender gender;  
6  
7     private int age;  
8  
9     // all args constructor & getters...  
10 }
```

- Как ще изглежда функция, която филтрира в един списък хората на възраст под 25 години?

# Пример: филтриране на списък от хора

- Функция, която филтрира в един списък хората, които са под определена възраст:

```
1 public static List<Person> filterByAgeLessThan(List<Person> people, int age) {
2     List<Person> result = new ArrayList<>();
3     for (Person person : people) {
4         if (person.getAge() < age) {
5             result.add(person);
6         }
7     }
8     return result;
9 }
```

- А как ще изглежда функция, която филтрира хората на определена възраст и от определен пол?

# Пример: филтриране на списък от хора

- Функция филтрираща по още един критерий:

```
1 public static List<Person> filterByAgeAndGender(List<Per  
2 List<Person> result = new ArrayList<>());  
3 for (Person person : people) {  
4     if (person.getAge() == age && person.getGender() ==  
5         result.add(person);  
6     }  
7 }  
8 return result;  
9 }
```

- Какви са разликите с предишната функция? Как може еднаквите части да се изнесат на едно място?

# Решение преди Java 8

- Добавя се параметър на филтриращата функция от абстрактен тип, в чиито абстрактни методи да се дефинира при извикване поведението, което искаме да се променя динамично

```
1 public interface Filter {  
2     boolean matches(Person person);  
3 }  
  
1 public static List<Person> filter(List<Person> people,  
2     Filter filter) {  
3     List<Person> result = new ArrayList<>();  
4     for (Person person : people) {  
5         if (filter.matches(person)) {  
6             result.add(person);  
7         }  
8     }  
9     return result;  
10 }
```

# Решение преди Java 8

- Извикването на функцията обикновено (но не задължително) става с инстанцирането на анонимен клас:

```
1 List<Person> people = Arrays.asList(  
2     new Person("Ivan", Gender.MALE, 22),  
3     new Person("Ivanka", Gender.FEMALE, 34));  
4  
5 List<Person> result = filter(people, new Filter() {  
6  
7     public boolean matches(Person person) {  
8         return person.getAge() > 12 && person.getAge() < 65  
9             && person.getGender() == Gender.FEMALE;  
10    }  
11 }); // ще съдържа само инстанцията с име "Ivanka"
```

- Но този синтаксис е неудобен, дълъг и по-трудно четим...

# Java 8 синтаксис

- Затова в Java 8 той е опростен чрез т. нар. ламбда функции - функции подадени като аргумент на други функции.

```
1 List<Person> result = filter(people,
2     person -> person.getAge() > 12
3         && person.getAge() < 65
4         && person.getGender() == Gender.FEMALE)
5 }); // ще съдържа само инстанцията с име "Ivanka"
```

- Декларацията на функцията `filter` от примера остава същата
- Резултатът от операцията в ламбда функцията се връща като резултат при извикването на `matches` (върху параметъра от тип `Filter`) в имплементацията на `filter`



# Функционални интерфейси

- Интерфейсът `Filter` наричаме функционален
- Функционален интерфейс е този, в който има точно един абстрактен метод
- За да може да се прилага този синтаксис, аргументът, на който се подава ламбда функцията, трябва да бъде инстанция на "функционален интерфейс"
- При абстрактни класове (дори и само с един абстрактен метод) или интерфейси с повече от един абстрактен метод, този синтаксис не може да се използва (в тези случаи могат да се използват анонимни класове)

# Функционални интерфейси

- В стандартната библиотека в пакета `java.util.function` има дефинирани множество шаблонни функционални интерфейси, които се използват както в самата библиотеката, така и могат да се използват от всеки програмист.
- `Function` - дефинира метод, който приема един шаблонен аргумент и връща шаблонен резултат
- `Consumer` - метод с един шаблонен аргумент и връщащ **void**
- `Supplier` - метод без аргументи и връщащ шаблонен резултат
- `Predicate` - метод с един шаблонен аргумент и връщащ резултат от тип **boolean**
- Съществуват вариации на тези функционални методи, които дефинират същите методи, но с два аргумента и се казват съответно `BiFunction`, `BiConsumer` и т.н.

# Пример: сортиране на списък с Comparator

- В `java.util.List` има метод `sort` с единствен аргумент от тип `Comparator`, който е функционален интерфейс

```
1 public interface Comparator<T> {  
2     int compare(T o1, T o2);  
3 }
```

```
1 List<Person> people = Arrays.asList(  
2     new Person("Ivan", Gender.MALE, 22),  
3     new Person("Ivanka", Gender.FEMALE, 34),  
4     new Person("Peter", Gender.FEMALE, 11));  
5 people.sort((p1, p2) -> p1.getAge() - p2.getAge());
```

# Ламбда функция на много редове

- В предишните примери ламбда функциите се състояха от само един израз
- Следващият пример използва метода `forEach` в `java.util.List`, за да принтира всички елементи по определен начин като използва ламбда функция на много редове

```
1 people.forEach(person -> {  
2     if (person.getAge() > 65) {  
3         System.out.printf("%s, >65+\n", person.getNames());  
4     } else {  
5         System.out.printf("%s, %d\n", person.getNames(),  
6             person.getAge());  
7     }  
8 });
```