Functional Programming with Java 8

Mihail Stoynov

Part 1
Moving to the Stream API

Using stream()

Why type erasure is so annoying?

Peek() for debugging

Final not explicitly needed now

About.me

@night

Part of the BG-JUG

Trying to contribute to OpenJDK

@day

Part of **STY** (java & security & training)

-deliver trainings, write code

-perform penetration tests

Bulgarian Java User Group

- Want to learn some new technology?
- Want to brag you just learned a new tech?
- Come as a listener or presenter
- BGJUG Members are OpenJDK contributors
  - We plan to be far more involved in how OpenJDK is shaped
- http://java-bg.org/
- bg-jug@googlegroups.com

Functional Programming

A way to write code that focuses on the WHAT

And not the HOW

Small difference semantically, big - syntactically

Lambdas help you pass anonymous functions

which are actually anonymous classes

Lambdas are just a syntactic sugar

WHAT & HOW

```
// we tell the code what to do and how to do it
//imperative - saying how;
//mutable
//mutability is evil, because we might break it
//the for cycle is familiar, yet complex, i++ for example, many moving parts
public static int countCapitals(String sentence) {
    int result = 0;
    for(char c : sentence.toCharArray()) {
        if(c >= 'A' && c <= 'Z') {
            result++;
        }
    }
    return result;
}
```

WHAT only

```
//small difference syntactically, big - semantically
//we tell
//only what to do, the code does it - declarative code, we can be declarative without functional style, but
//functional is declarative
public static long countCapitalsLambda(final String sentence) {
    return sentence
                    .codePoints()
                    .filter(c -> c >= 'A' && c <= 'Z')
                    .count();
}
```

WHAT only

Mutability is evil, because we might break stuff

With functional style we can be declarative

- we can do it without lambdas
- but it's so much better with lambdas

The Stream API is just a fancy Strategy Pattern

- or dependency injection

Primitive types and generics

```
// I WANT TYPE REIFICATION (FIX GENERICS PLEASE)
// probably it'll happen in java 10. Probably...
// the following two lines are different
IntPredicate predicate = c -> c >= 'A' && c <= 'Z';
Predicate<Integer> p2 = c -> c >= 'A' && c <= 'Z';
```

final String sentence

```
//small difference syntactically, big – semantically
// we tell only what to do, the code does it - declarative
// code, we can be declarative without functional style,|
// but functional is declarative
public static long countCapitalsLambda(String sentence) {
    // with lambdas ‘sentence’ does not have to be final,
    // but you cannot edit it
    return sentence
            .codePoints()
            .filter(c -> c >= 'A' && c <= 'Z')
            .peek(i -> System.out.println(sentence+" "+i))
            .count();

}
```

Part 2
Optional and Method references

How cool is optional?

Method references

Lazy init

How cool is optional? (rhetorical)

```
// ‘Optional’ prevents NPEs
public static Optional<Character> findFirstCapital(String sentence) {
    for(char c : sentence.toCharArray()) {
        if(c>='A' && c<='Z') {
            return Optional.of(Character.valueOf(c));
        }
    }
    return Optional.empty();
}
```

Method references

```
// method reference is passing a method to
// something that expects lambda
OptionalInt firstCapitalWithLambdas =
        theSentence
                .codePoints()
                .filter(this::isCapitalLetter)
                .findFirst();
```

Method ref to variable

```
// can we do this? Yeah we can
Predicate<Integer> isEven =
        P2OptionalAndMethodReferences::isCapitalLetter;
```

Lazy initialization

map(), filter() - intermediate operations

findFirst(), reduce(), collect() – terminal ops

If you don't execute a terminal operation

nothing will happen

A terminal triggers all the rest

they were lazy before that

they wouldn’t do anything before that

Part 3
Higher-order functions, legacy code

HOF

Sending lambdas to legacy code

Higher-order functions

```
//duplication is fundamentally evil - let's unite 1 and 2
//sending functions to functions = higher-order functions
// (hof) (takes or returns a function)
//hof gives you lazy evaluation (may not evaluate the
// incoming func) & composition (transformation of objects
// instead of mutation of objects)

BiFunction<Character, Character, IntPredicate> isBetween =
                (a, z) -> (i -> i >= a && i <= z);
```

Sending lambdas to legacy code

```
// we are injecting lambdas, but the method takes a
// Chooser, so we're again
// using classes, but we're mixing object
// composition with function composition (no ceremony)
printChars(
                    theSentence,
                    (IntPredicate)i->Character.isUpperCase(i));
```

Q&A

Functional Programming with Java 8

Mihail Stoynov, STY

BG JUG