Java 8

λ Expressions

Scott
Leberknight

Mom, why?

Declare what to do, not how to do it

Promote immutability

Easier parallelization & lazy evaluation

Cleaner, more concise code

iterating w/ forEach

(under the hood)

```
List<Integer> numbers =
    Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

numbers.forEach(new Consumer<Integer>() {
    @Override
    public void accept(Integer integer) {
        System.out.println(integer);
    }
});
```

lambda expressions

λ

```
(argument list) -> code
```

examples:

```
(final String name, final int age) -> code

(String name, int age) -> code

(name, age) -> code

name -> code
```

} type inference

iterating w/lambdas

```
List<Integer> numbers =
    Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

numbers.forEach((final Integer value) -> System.out.println(value));
```

lambda expression

iterating w/lambdas, type inference

```
List<Integer> numbers =
    Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

numbers.forEach(value -> System.out.println(value));
```

method references

```
List<Integer> numbers =
    Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

numbers.forEach(System.out::println);
```

method reference

transforms using map

```
List<Integer> numbers =
    Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

numbers.stream()
    .map(value -> value * 2)
    .forEach(System.out::println);
```

filtering

```
List<Integer> numbers =
    Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

numbers.stream()
    .filter(value -> value % 2 == 0)
    .forEach(System.out::println);
```

filtering v2

```
List<String> names = Arrays.asList(
  "Bob", "Tom", "Jeff", "Scott", "Jennifer", "Steve");

final Function<String, Predicate<String>> startsWithLetter =
        letter -> name -> name.startsWith(letter);

names.stream()
        .filter(startsWithLetter.apply("J"))
        .forEach(System.out::println);
```

Streams

```
Streams.iterate(1, number -> number + 1)
    .map(number -> number * number)
    .limit(25)
    .forEach(number -> System.out.print(number + " "));
// 1 4 9 16 25 36 49 64 ... 529 576 625
```

finding with Optional

```
List<String> names = Arrays.asList(
  "Bob", "Tom", "Jeff", "Scott", "Jennifer", "Steve");

Optional<String> firstS = names.stream()
    .filter(name -> name.startsWith("S"))
    .findFirst();
System.out.println(firstS.orElse("None found"));
```

reducers

```
List<String> names = Arrays.asList(
    "Bob", "Tom", "Jeff", "Scott", "Jennifer", "Steve");

Optional<String> longestName = names.stream()
    .reduce((name1, name2) ->
        name1.length() >= name2.length() ? name1 : name2);

if (longestName.isPresent()) {
    System.out.println(longestName.get());
}
```

sum reducer

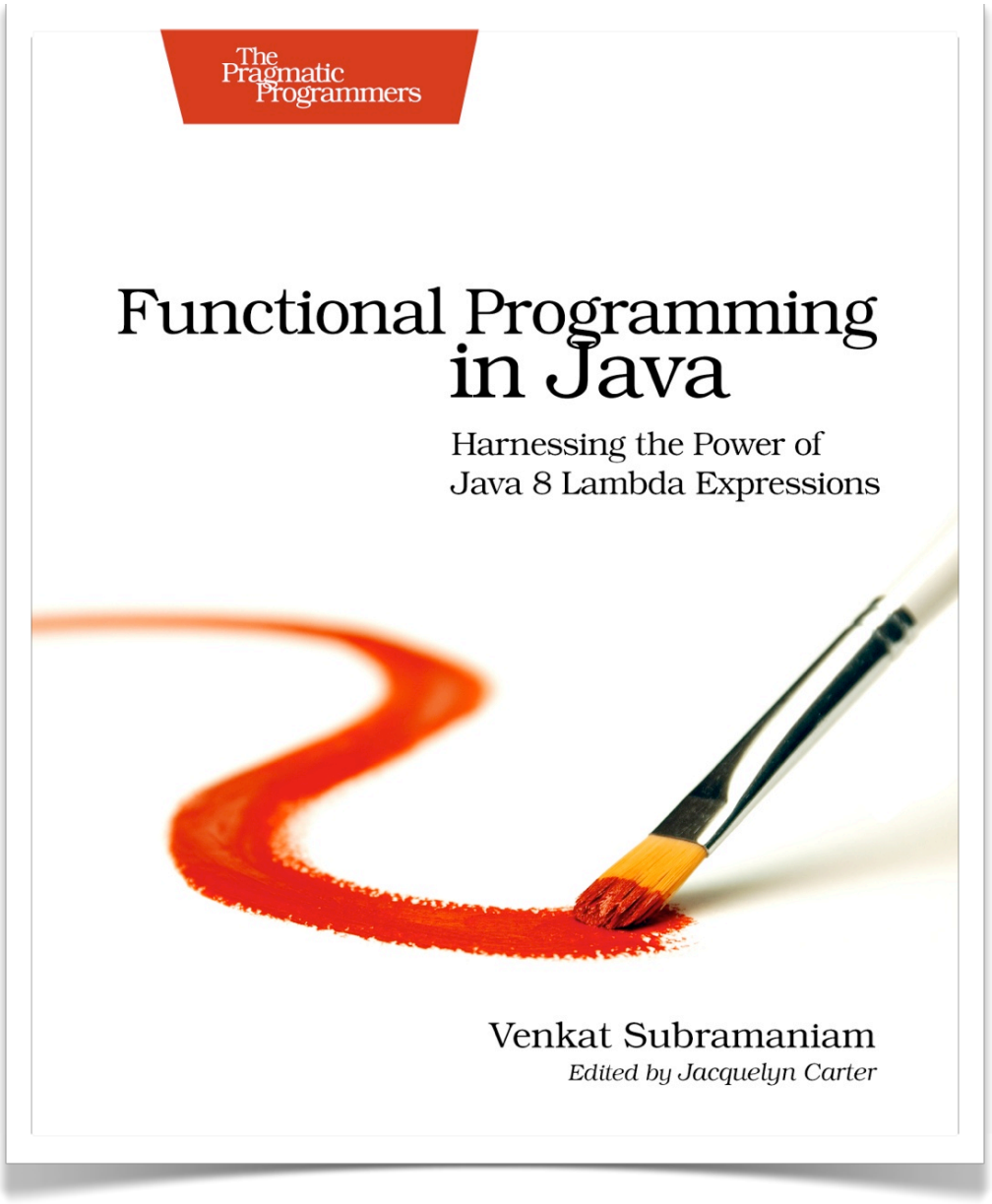
```
List<Integer> numbers =
    Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

long sum = numbers.stream()
    .map(value -> value * value)
    .sum();
```

parallelization

```
numbers.parallelStream()
       .filter(number -> isPerfect(number))
       .forEach(System.out::println);
```


how far we went in this short talk...



http://pragprog.com/book/vsjava8/functional-programming-in-java

JDK 8 Project

http://jdk8.java.net/

sample code available at:

https://github.com/sleberknight/java8-lambda-samples

My Info

scott dot leberknight at nearinfinity dot com

scott dot leberknight at gmail dot com

twitter.com/sleberknight

www.sleberknight.com/blog

www.nearinfinity.com/blogs/scott_leberknight/all/