Reaching the lambda heaven

Víctor Orozco

February 23, 2017

Nabenik

About

NABENIK

Student 1 - How do I **jump** the CS theory to do "real development" with Java 8?

Street developer 1 - How do I **jump** the CS theory to do "real development" with Java 8?

$$
\begin{aligned}
x[t/x] &= t \\
y[t/x] &= y \text{ if } x \neq y \\
c[t/x] &= c \\
(s_1\ s_2)[t/x] &= s_1[t/x]\ s_2[t/x] \\
(\lambda x.\ s)[t/x] &= \lambda x.\ s \\
(\lambda y.\ s)[t/x] &= \lambda y.\ (s[t/x]) \text{ if } x \neq y \text{ and either } x \notin FV(s) \text{ or } y \notin FV(t) \\
(\lambda y.\ s)[t/x] &= \lambda z.\ (s[z/y][t/x]) \text{ otherwise, where } z \notin FV(s) \cup FV(t)
\end{aligned}
$$

The only difference is in the last two lines. We substitute as before in the two safe situations where either x isn't free in s, so the substitution is trivial, or where y isn't free in t, so variable capture won't occur (at this level). However where these conditions fail, we first rename y to a new variable z, chosen not to be free in either s or t, then proceed as before. For definiteness, the variable z can be chosen in some canonical way, e.g. the lexicographically first name not occurring as a free variable in either s or t.[3]

How do I **learn** FP to do "street development" with Java 8?

What should I **learn** FP to do "street development" with Java 8?

Outline

Java 8

FP

Functional blocks

Functional JDK

Libraries

QA

Java 8

Java 8

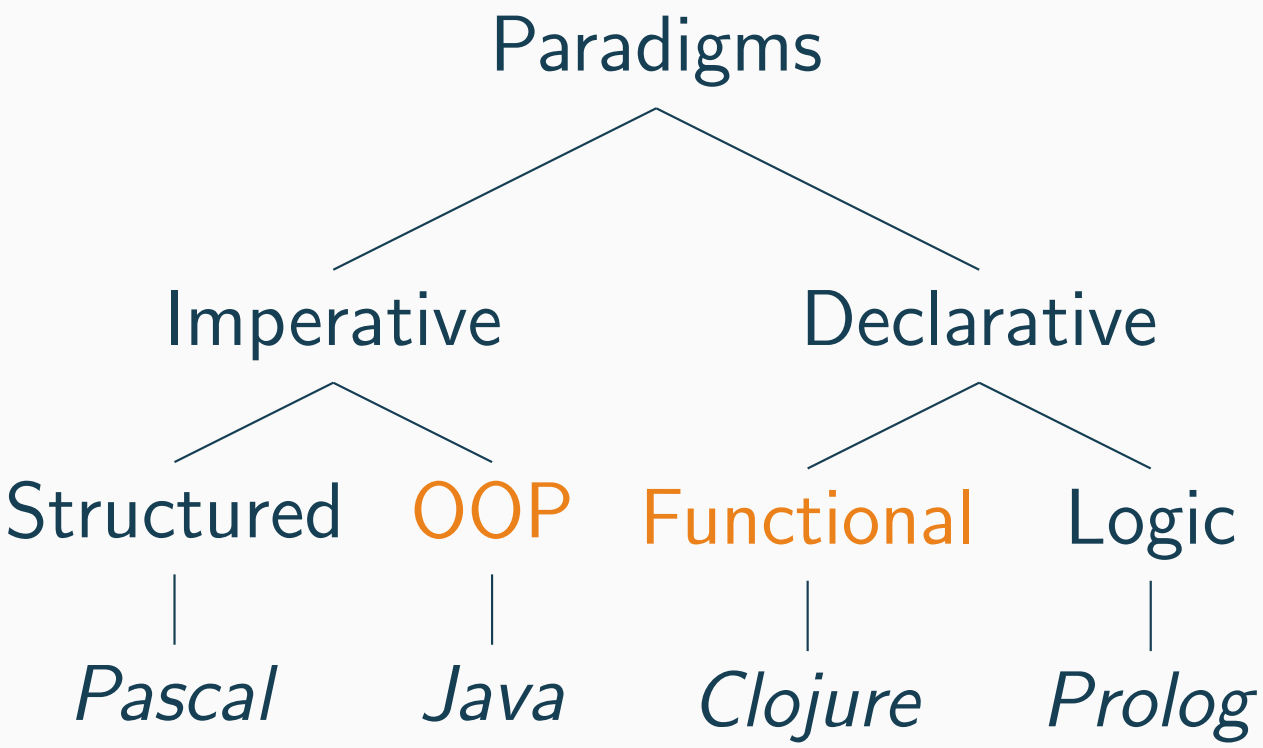
Release date: 2014-03-18 - 3 years ago!!

https://www.oracle.com/java8
https://www.oracle.com/java8launch

Java 8

- Nashorn
- Date/Time API
- Compact Profiles
- Type Annotations
- **Default methods**
- **Streams**
- **Lambda Expressions**

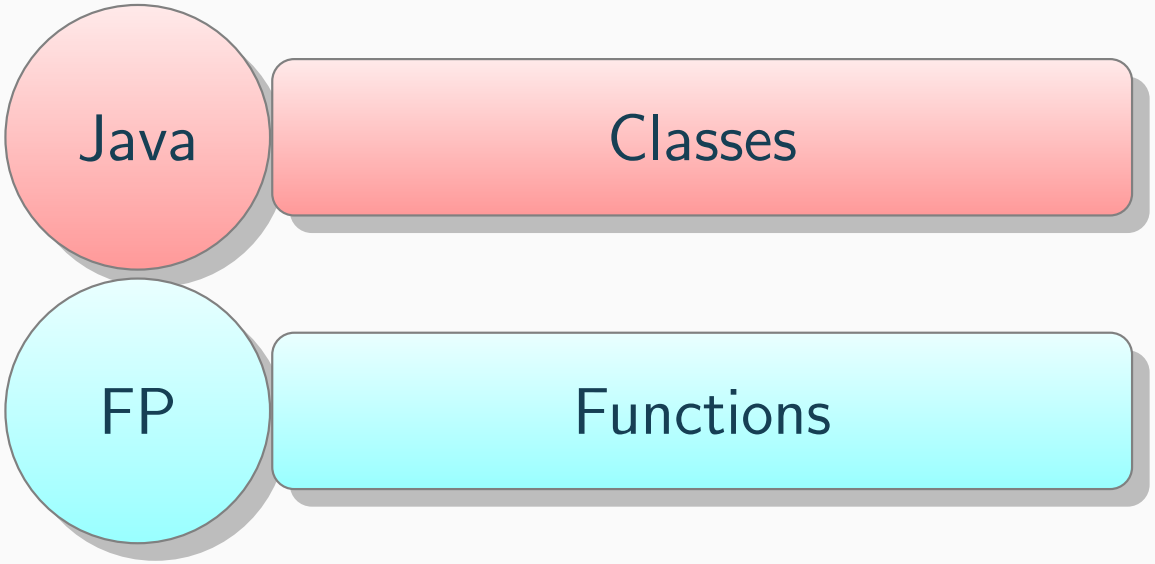
Paradigms (Simplification)



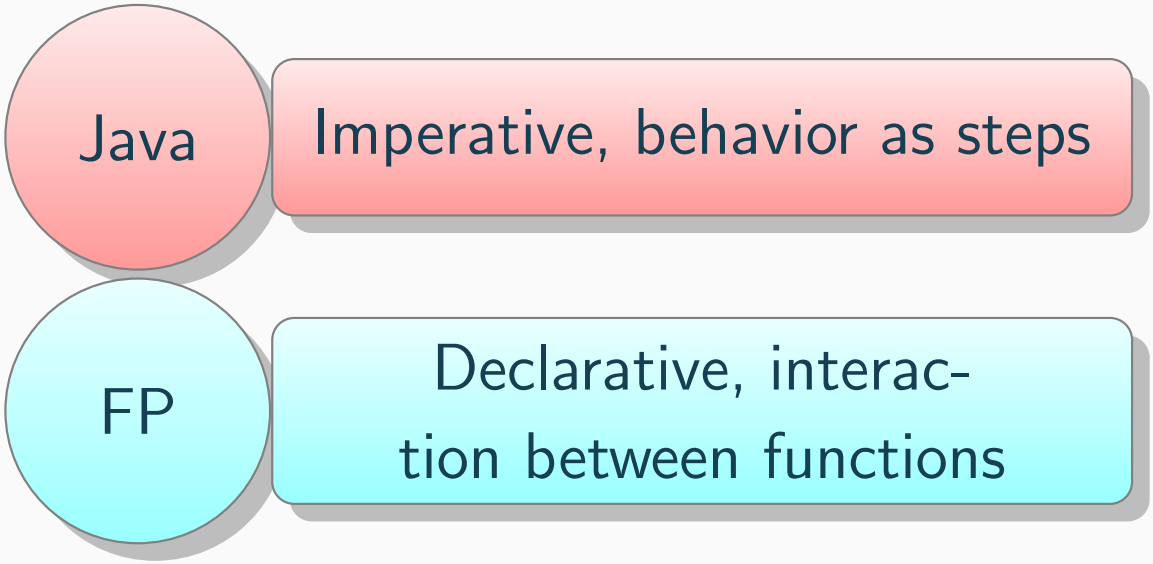
1-2-3-4 of FP

1. Computation = Function evaluation
2. NO state changes
3. NO mutability
4. Declarative $\rightarrow$ Expressions

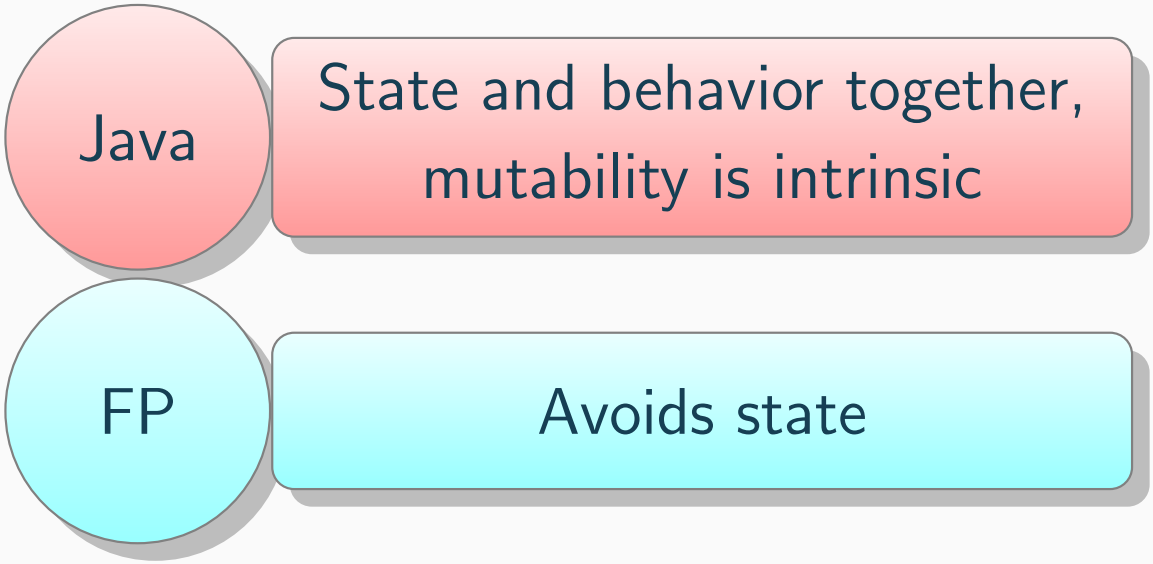
Java vs. Functional (organization - think about)



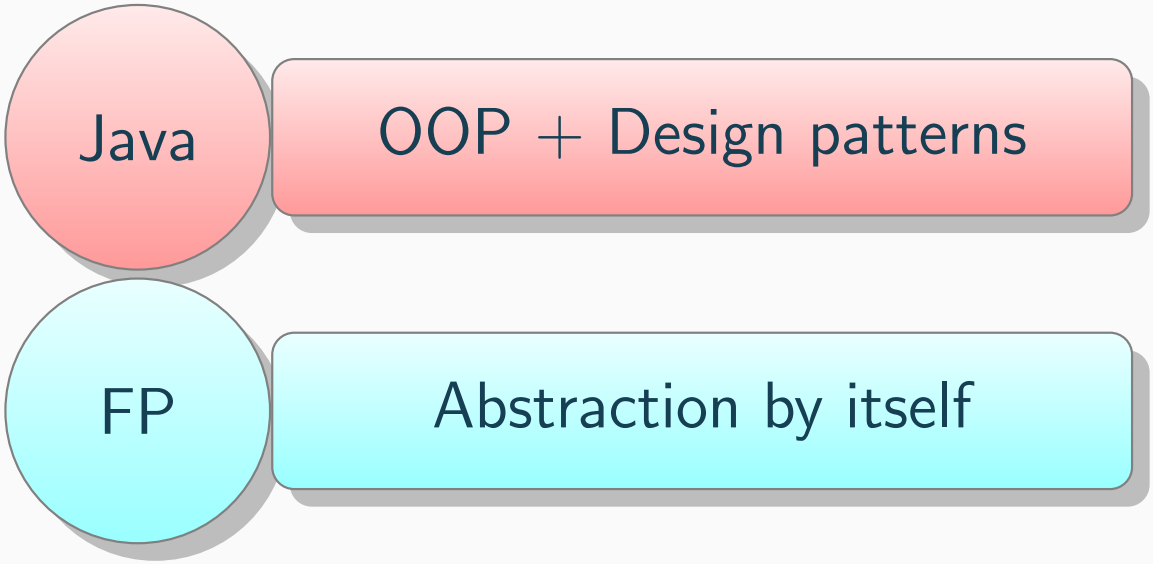
Java vs. Functional (algorithms - write code as)



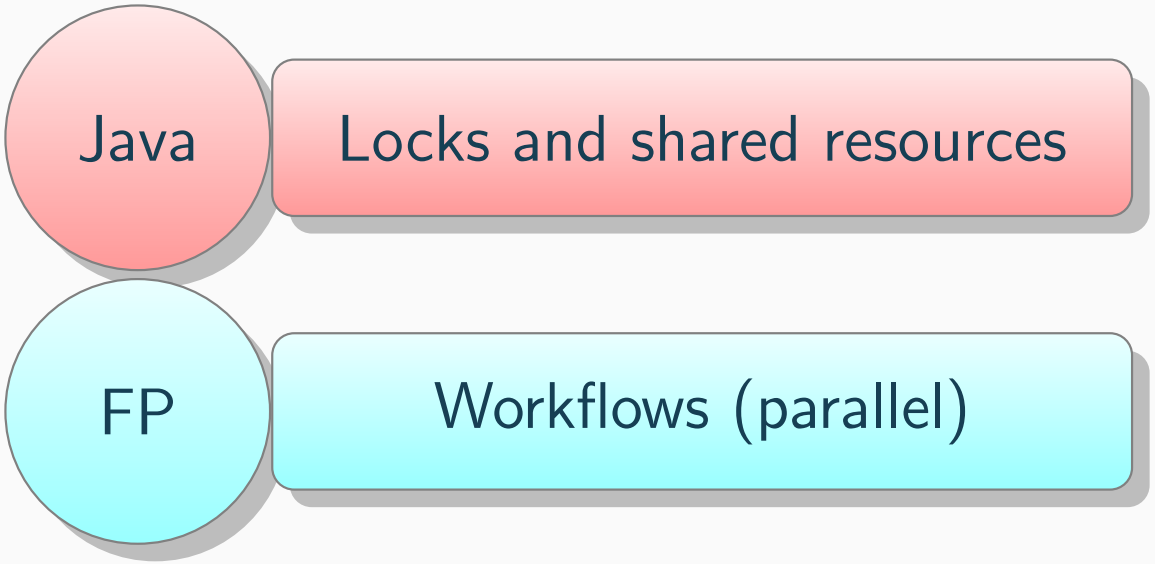
Java vs. Functional (Mutability and state - manipulate or not)



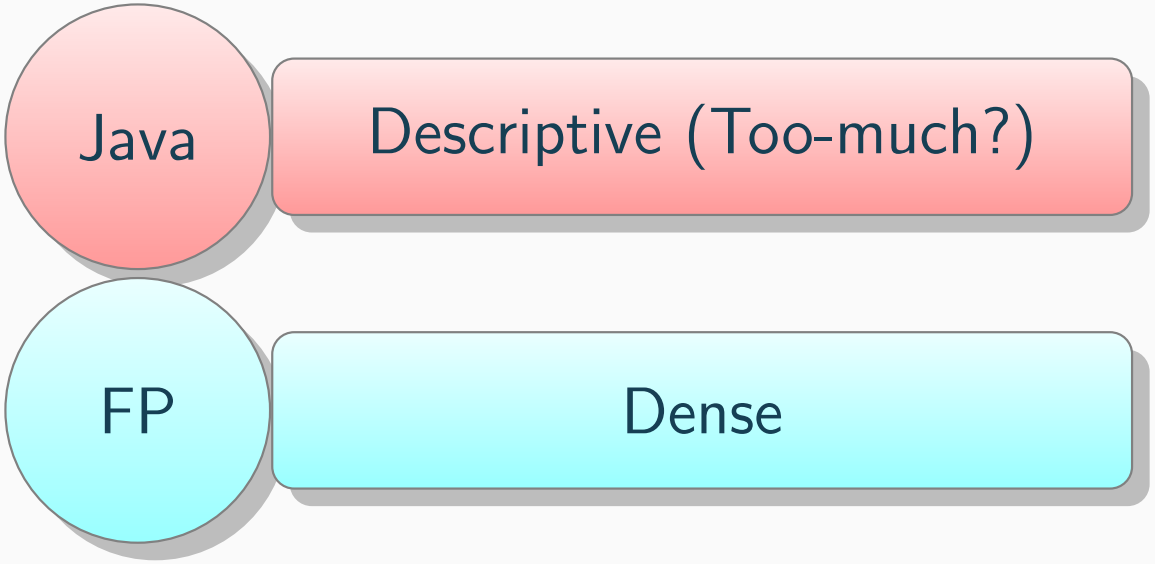
Java vs. Functional (Style - Good looking code)



Java vs. Functional (Concurrency - The most hated thing in the development world)



Java vs. Functional (Code - Result)



Java 8

An OOP language with functional additions

FP

Why?

1. Easy parallelism
2. Elegance
3. Good with reactive

FP on Java

- Java is not purely functional
- Other options (Scala, Kotlin, Ceylon)
- Java supports FP through libraries

FP on Java (Consequence)

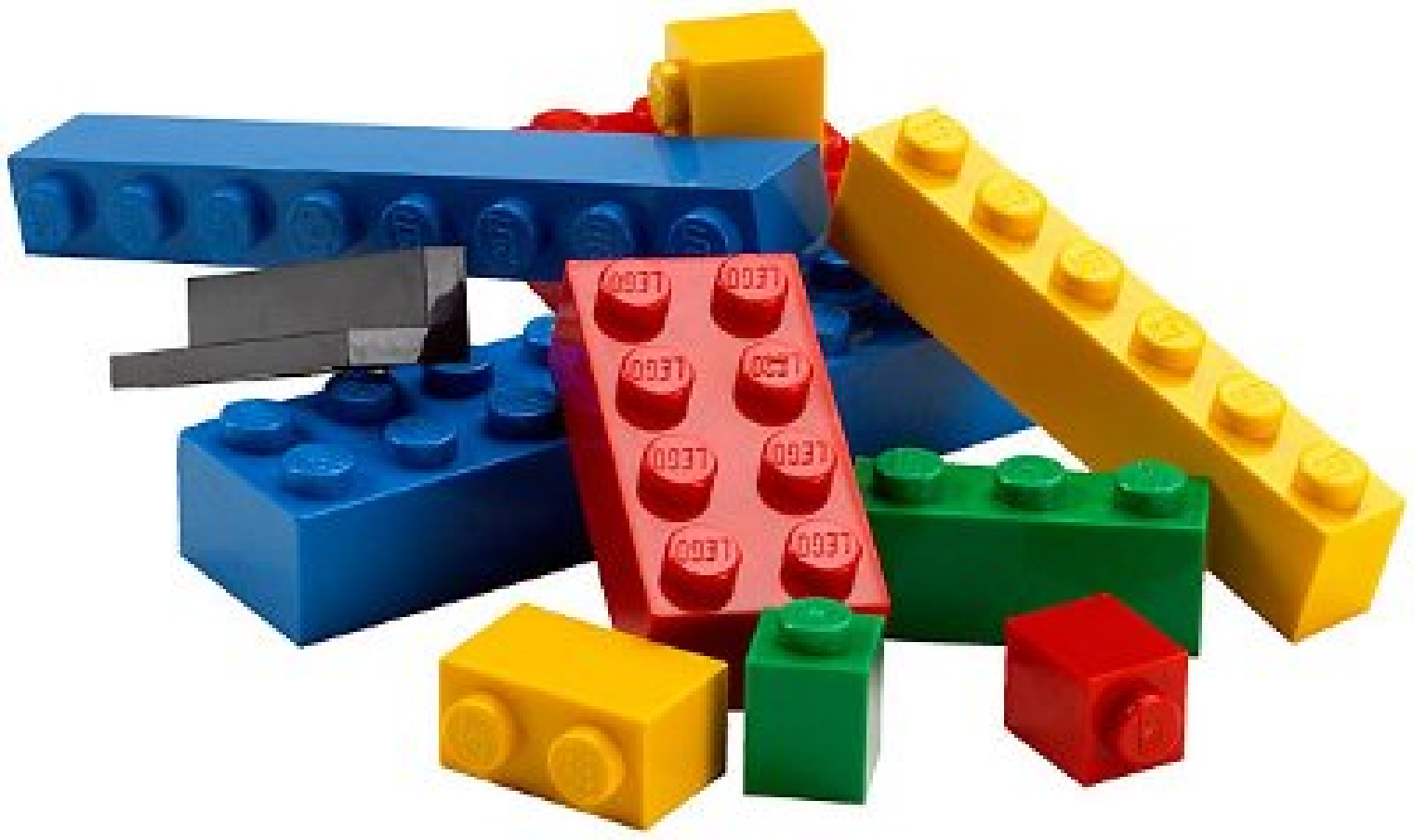
FP on Java (Consequence)

Java 8 spawned a new ecosystem of functional (and declarative . . . and reactive) libraries

Q - How do I reach it?

A - Using a starway

1 - Blocks



1 - Functional blocks in Java 8

- Lambda expressions
- Functional interface
- High order functions
- Complements (predicates, method reference)

2 - The JDK

2 - Functional JDK

- Pre-defined functions in Java 8 (java.util.function)
- More than 40 functional interfaces
- Streams API

3 - Libraries

3 - Libraries

Cathedral

- Java EE (Reactive on the road)
- Spring (FP Ready)
- Lagom (Reactive oriented)

Bazaar

- JavaSlang, jOOQ/jOOL, EclipseCollections, FunctionalJava (Functionality)
- RxJava, Akka, Vert.x (Interactions - Architecture)

Functional blocks

Lambda expression

Anonymous function (behavior)

```
//In-line
(String x) -> System.out.println(x);

//Multi-line
(x) -> {
    System.out.println(x);
}

//From static
System.out::println;
```

Functional interfaces

- Just one abstract method
- Interfaces allow default methods

```
@FunctionalInterface
public interface MyFunctionalInterface
{
        String doFunctional(String a, String b);
}
```

High order functions

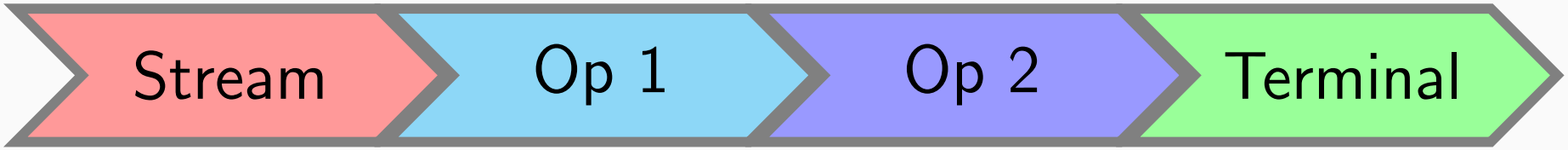
- Functions as arguments and return values

```
MyFunctionalInterface doHoFunction
    (MyfunctionalInterface param){
    String result = param.doFunctional(
        "Marco", "Polo");
    return (x,y) -> x.concat(y);
}
```

Functional JDK

Streams API

- Map-Reduce
- "Monad" like



Streams API

Declarative - Initial

```
List<String> goodJvmLangs =
        Arrays.asList("Java", "Kotlin", "Ceylon");

//Method reference
goodJvmLangs.forEach(System.out::println);
```

Streams API - Predicates

```
List<String> goodJvmLangs =
        Arrays.asList("Java", "Kotlin", "C#");

Predicate<String> badRemover =
        s -> "C#".equals(s);

goodJvmLangs.removeIf(badRemover);
```

Streams API - Map-reduce

```
winners
        .stream() //Steam
        .filter( //Intermediate
                p -> p.getOrderId()
                        .equals(winnerId)
                )
        .findFirst(); //Reduce
```

Streams API - Map-reduce

```
unfilteredList.stream()
    .map(x -> x-1) //Real ap
    .filter(x -> x > 50) //Other intermediate app
    .collect(Collectors.toList()); //Reduce
```

Libraries

JavaSlang

Offer

- Java core library
- Immutable collections
- Control structures

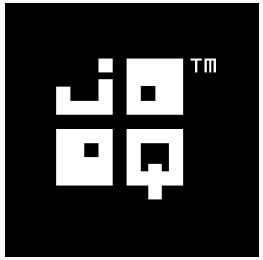
Good for

- Elegant code
- Eliminating the *.stream*() and *.collect*() in streams
- Exception handling (Try monad)

jOOQ

Offer

- Database first
- Typesafe SQL
- Code generation



Good for

- Natural SQL queries
- Low overhead queries
- Stream processing of DB results

Vert.x

Offer

- Reactive tool-kit
- Modular
- Scalable

VERT.X

Good for

- Reactive backend and/or microservices
- Compatible with RxJava
- Alternative framework

Complete sample

http://github.com/tuxtor/fpjavademo2

Bazaar caveats

- Many library features overlap (Cyclops - `https://github.com/aol/cyclops`)
- Streams API improvements in Java 9 `https://www.voxxed.com/blog/2017/02/java-9-streams-api/`
- Huge POM.xml :)

FP - The good parts

- Fun
- Declarative
- Less and elegant code

FP - The bad parts

- Performance - (maybe)
- Debug
- Learning curve

Books and resources

- JDK 8 Lamdas&Streams MOOC https://www.youtube.com/playlist?list=PLMod1hYiIvSZL1xclvHcsV2dMiminf19x
- Functional Programming in Java: Harnessing the Power Of Java 8 Lambda Expressions http://www.amazon.com/Functional-Programming-Java-Harnessing-Expressions/dp/1937785467

QA

Thank you

- @tuxtor
- me@vorozco.com
- http://vorozco.com
- http://github.com/tuxtor/slides