Functional Programming without Lambdas

Johan Haleby
Mattias Severson

Agenda

- Intro
- Example
  - Functional Java
  - LambdaJ
  - Guava
- Summary

JAYWAY

Ubiquitous

python™

Lisp

JavaScript

Ubiquitous

JAYWAY

Survey

- Java 8?
- Java 7?
- Java 6?
- Java <= 5?

JAYWAY

For Legacy Java?

functional. JAVA

Apache Commons™
http://commons.apache.org/

f(x) Totally Lazy

op4j

f(u)ncito

JAYWAY

Focus on what, not how!

```
for (int i=0; i<persons.size(); i++) {
    System.out.println(persons.get(i));
}
```

```
persons.foreach(println)
```

JAYWAY

Map


```
return beer.getPrice();
```

JAYWAY

Filter


```
return beer.getPrice() > 20;
```

JAYWAY

Reduce / Fold


```
return a + b;
```

JAYWAY

The Receipt...

Systembolaget
Rålambsvägen 7-9
S-112 59 STOCKHOLM
08/13 30 95
OrgNr: 556059-9473

Förs : 0132osja But: 0132 Nr: 1406
Datum : 2011-04-23 14:27 Ka: 3

Staropramen Granat 4,8% 330 ml
1609-03 15,90
Sofiero Original 5,2% brk 500 ml
1222-02 4 * 10,90 43,60
Red Stripe 4,7% 330 ml
1531-03 14,90
TT 4,5% 330 ml
1352-03 10,90
Törley Charmant Rosé 750 ml
7713-01 54,00
Oppigårds Golden Ale 5,2% 330 ml
1490-03 17,50
Sierra Nevada Pale Ale 5,6% 355ml
1525-03 22,90
Staropramen Granat 4,8% 330 ml
1609-03 15,90
Sofiero Original 5,2% brk 500 ml
1222-02 4 * 10,90 43,60
Red Stripe 4,7% 330 ml
1531-03 14,90
TT 4,5% 330 ml
1352-03 10,90
Törley Charmant Rosé 750 ml
7713-01 54,00
Oppigårds Golden Ale 5,2% 330 ml
1490-03 17,50
Sierra Nevada Pale Ale 5,6% 355ml
1525-03 22,90
Staropramen Granat 4,8% 330 ml
1609-03 15,90
Sofiero Original 5,2% brk 500 ml
1222-02 4 * 10,90 43,60
Red Stripe 4,7% 330 ml
1531-03 14,90
TT 4,5% 330 ml
1352-03 10,90
Oppigårds Golden Ale 5,2% 330 ml
1490-03 17,50
Sierra Nevada Pale Ale 5,6% 355ml
1525-03 22,90
Sierra Nevada Pale Ale 5,6% 355ml
1525-03 22,90

TOTAL [illegible]

Beer

```
public interface Beer {

    String getName();
    Type getType();
    int getPrice();

}
```

JAYWAY

Use Cases

- Price of cheapest beer
- Price of most expensive beer
- Name of the most expensive beer
- Names of all Lagers
- Total price

JAYWAY

Functional Java

```
public abstract class F<A, B>() {
    public abstract B f(A a);
}
```

Cheapest Beer

```
List<Beer> beers = receipt.getBeers();

int minPrice = fj.data.List.iterableList(beers).map(
    new F<Beer, Integer>() {
        @Override
        public Integer f(Beer beer) {
            return beer.getPrice();
        }
}).minimum(Ord.intOrd);
```

Cheapest Beer

```
static final F<Beer,Integer> BEER_PRICE = new F<Beer,Integer>() {
        @Override
        public Integer f(Beer beer) {
            return beer.getPrice();
        }
    };
```

Cheapest Beer

```
public static <T> fj.data.List<T> with(Iterable<T> it) {
    return fj.data.List.iterableList(it);
}
```

Cheapest Beer

```
int minPrice = with(beers).map(BEER_PRICE).minimum(Ord.intOrd);
```

Most Expensive Beer

```
int maxPrice = with(beers).map(BEER_PRICE).maximum(Ord.intOrd);
```

Name of Most Expensive Beer

```
String name = with(beers).maximum(ord(BEER_PRICE_ORDER)).getName();
```

Name of Most Expensive Beer

```
public static final F<Beer, F<Beer, Ordering>> BEER_PRICE_ORDER =
new F<Beer, F<Beer, Ordering>>() {
    @Override
    public F<Beer, Ordering> f(final Beer beer1) {
        return new F<Beer, Ordering>() {
            @Override
            public Ordering f(Beer beer2) {
                Integer beer1Price = beer1.getPrice();
                Integer beer2Price = beer2.getPrice();

                if (beer1Price > beer2Price) {
                    return Ordering.GT;
                } else if (beer1Price < beer2Price) {
                    return Ordering.LT;
                } else {
                    return Ordering.EQ;
                }
            }
        };
    }
};
```

Names of Lager Beers

```
Collection<String> names =
with(beers).filter(type(LAGER)).map(BEER_NAME).toCollection();
```

Names of Lager Beers

Collection with

```
public static F<Beer, Boolean> type(final Type type) {
    return new F<Beer, Boolean>() {
        @Override
        public Boolean f(Beer beer) {
            return beer.getType().equals(type);
        }
    };
}
```

Names of Lager Beers

Collection
with

```
static final F<Beer,String> BEER_NAME = new F<Beer,String>() {
        @Override
        public String f(Beer beer) {
            return beer.getName();
        }
    };
```

Names of Lager Beers

```
Collection<String> names =
with(beers).filter(type(LAGER)).map(BEER_NAME).toCollection();
```

[Heineken, Heineken, Carlsberg, Carlsberg, Heineken, Heineken, Carlsberg, Heineken, Heineken, ...]

Names of Lager Beers

```
Collection<String> names =
with(beers).filter(type(LAGER)).map(BEER_NAME).nub().toCollection()
```

[Carlsberg, Heineken]

Sum

```
int sum = with(beers).map(BEER_PRICE).foldLeft(SUM, 0);
```

Sum

`int`

```
public static final F2<Integer,Integer,Integer> SUM =
    new F2<Integer, Integer, Integer>() {
        @Override
        public Integer f(Integer accSum, Integer beerPrice) {
            return accSum + beerPrice;
        }
    };
```

functional. JλVλ

JAYWAY

Sum

```
int sum = with(beers).map(BEER_PRICE).foldLeft1(Integers.add);
```

Functional Java

+ Naming conventions
+ Powerful
+ Parallelism
+ Good learning platform
+ Immutable

Functional Java

- Non-standard collections
- Inconsistent API
- Powerful
- Not always obvious (_.1())

Functional Java Madness

```
/**
 * Returns a function that returns the eighth element of a product.
 *
 * @return A function that returns the eighth element of a product.
 */
public static <A, B, C, D, E, F$, G, H> fj.F<P8<A, B, C, D, E, F$, G, H>, H> __8(){
  return new fj.F<P8<A, B, C, D, E, F$, G, H>, H>() {
    public H f(final P8<A, B, C, D, E, F$, G, H> p) {
      return p._8();
    }
  };
}
```

LambdaJ

- Implicit function creation

```
Lambda.on(Beer.class).getPrice()
```

JAYWAY

Cheapest Beer

```
int min = with(beers).min(on(Beer.class).getPrice());
```

Cheapest Beer

```
int min = with(beers).min(on(Beer.class).getPrice());
```

Most Expensive Beer

```
int max = with(beers).max(on(Beer.class).getPrice());
```

Name of Most Expensive Beer

```
String name =
    with(beers).selectMax(on(Beer.class).getPrice()).getName();
```

JAYWAY

Name of Lager Beers

```
List<String> names =
    with(beers).
        retain(having(on(Beer.class).getType(), is(LAGER))).
        extract(on(Beer.class).getName());
```

[Heineken, Heineken, Carlsberg, Carlsberg, Heineken, Heineken, Carlsberg, Heineken, Heineken, ...]

JAYWAY

Name of Lager Beers

```
Set<String> names =
    with(beers).
        retain(having(on(Beer.class).getType(), is(LAGER))).
        extract(on(Beer.class).getName()).
        distinct();
```

[Heineken, Carlsberg]

JAYWAY

Sum

```
int sum =
    with(beers).extract(on(Beer.class).getPrice()).
        aggregate(
            new PairAggregator<Integer>() {
                @Override
                Integer emptyItem() {
                    return 0;
                }

                @Override
                Integer aggregate(Integer accSum, Integer price) {
                    return accSum + price;
                }
            });
```

JAYWAY

Sum

```
int sum = with(beers).sum(on(Beer.class).getPrice());
```

LambdaJ

+ Less code
+ Standard Collections
+ Supports the “basic stuff”
+ “Closures”

LambdaJ

- Final classes
- Naming
- Slower than native
- Not Lazy
- Hard to read (possibly)
- Latest release not in Maven Central (issue 91)

Guava

```
public interface Function<F, T> {
  T apply(F input);
}
```

```
public interface Predicate<T> {
    boolean apply(T input);
}
```

JAYWAY

Cheapest Beer

```
int min = from(beers).
    transform(BEER_PRICE).
    toSortedImmutableList(Ordering.natural()).
    get(0);
```

JAYWAY

Cheapest Beer

```
int
```

```
static final Function<Beer,Integer> BEER_PRICE =
    new Function<Beer, Integer>() {
        @Override
        public Integer apply(Beer beer) {
            return beer.getPrice();
        }
    };
```

JAYWAY

Most Expensive Beer

```
int max =
    from(beers).
    transform(BEER_PRICE).
    toSortedImmutableList(Ordering.natural().reverse()).
    get(0);
```

JAYWAY

Name of Most Expensive Beer

```
String name =
    Ordering.from(BEER_PRICE_COMPARATOR).max(beers).getName();
```

JAYWAY

Name of Lagers

```
Iterable<String> names =
    from(beers).filter(type(LAGER)).transform(BEER_NAME);
```

JAYWAY

Name of Lagers

```
Iterable<String> names =
    from(beers).filter(type(LAGER)).transform(BEER_NAME);
```

```
private static Predicate<Beer> type(final Type type) {
    return new Predicate<Beer>() {
        @Override
        public boolean apply(Beer beer) {
            return beer.getType().equals(type);
        }
    };
}
```

JAYWAY

Name of Lagers

```
Iterable<String> names =
    from(beers).filter(type(LAGER)).transform(BEER_NAME);
```

[Heineken, Heineken, Carlsberg, Carlsberg, Heineken, Heineken, Carlsberg, Heineken, Heineken, ...]

JAYWAY

Name of Lagers

```
Set<String> names =
    from(beers).filter(type(LAGER)).transform(BEER_NAME).
    toImmutableSet();
```

[Heineken, Carlsberg]

JAYWAY

Sum

- Not possible!!
  - Issue 218

JAYWAY

Guava

+ You may already use it
+ filter is called *filter*
+ Standard Collections
+ Lazy

JAYWAY

Guava

- Bloated
- Naming
- No Reduce / Fold
- Confusing entry-points

JAYWAY

Summary

- Functional Programming is here!
- Start looking into it, today!
- There are frameworks for Java

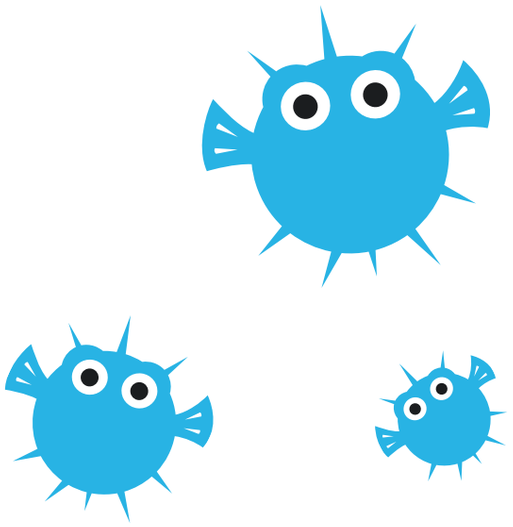
JAYWAY

Disclaimer

> when you go to preposterous lengths to make your code "a one-liner," the Guava team weeps.

JAYWAY

Questions?



Parallel

```
ExecutorService es = Executors.newFixedThreadPool(4);

Strategy<Integer> strategy = Strategy.executorStrategy(es);

int sum = strategy.parMap(BEER_PRICE, with(beers))
    ._1().foldLeft1(Integers.add);
```

Funcito

- Wraps Java methods as the function-type objects
- Works with existing frameworks

Example: FJ Sum

```
int sum = with(beers).map(BEER_PRICE).foldLeft1(Integers.add);


F<Beer,Integer> BEER_PRICE = new F<Beer,Integer>() {
    @Override
    public Integer f(Beer beer) {
        return beer.getPrice();
    }
};


F<Beer, Integer> BEER_PRICE = fFor(callsTo(Beer.class).getPrice());
```

f(u)ncito

JAYWAY

Thank You