# Les concepts de la programmation fonctionnelle illustrés avec Java 8

by Yannick Chartois
@ychartois

# Bodil Stokke

## What Every Hipster Should Know About Functional Programming

Vimeo / Parleys

# Question

Maintenant que nous avons les lambdas, peut-on faire la même chose avec Java 8?

# First Class Function

**Définition**

*“Les fonctions sont traitées par le langage comme des valeurs de première classe.”*

## Code

```
Function<String, String> hello = (String s) -> "hello " + s;
```

## Résultat

```
hello ;
// BaseConcepts$$Lambda$1@a09ee92
// != BaseConcepts@30f39991

hello.apply("Erouan") ;
// hello Erouan
```

# Higher order function

**Définition**

*“C'est une fonction qui prend une ou plusieurs fonctions comme entrée et/ou qui renvoie une fonction”*

# Filter

## Code

```
List<String> filter( Predicate<String> f, List<String> values ) {
      List<String> toReturn = new ArrayList<>();
      for( String current : values ) {
            if ( f.test(current) )
                  toReturn.add( current );
      }
      return toReturn; }
```

## Résultat

```
List<String> confs = Arrays.asList("jug", "devoxx", "javaone");
filter(s -> s.contains("j"), confs) ;
// [jug, javaone]
```

## Java 8

```
confs.stream().filter( s -> s.contains("j") ).collect(Collectors.toList())
```

# Functor

**Définition**

*“C'est une collection d'éléments X qui peut s'appliquer une fonction f: X -> Y pour créer une collection Y”*

# Map

## Code

```
List<String> map( Function<String, String> f , List<String> values ) {
      List<String> toReturn = new ArrayList<>();
      for( String current : values ) {
            toReturn.add( f.apply(current) );
      }
      return toReturn; }
```

## Résultat

```
List<String> confs = Arrays.asList( "jug", "devoxx", "javaone" );
map( s -> s.toUpperCase(), confs );
// [JUG, DEVOXX, JAVAONE]
```

## Java 8

```
confs.stream().map( s -> s.toUpperCase() ).collect( Collectors.toList() )
```

# Reduction / Aggregation

**Définition**

*Higher order function dont le but est de produire une valeur qui est le résultat de l'application d'un opérateur sur tous les éléments d'une structure de donnée*

# Reduce / fold

## Code

```
String reduce( BinaryOperator<String> op , List<String> values ) {
      String toReturn = "";
      for( String current : values ) {
            toReturn = toReturn.isEmpty() ? current : op.apply(toReturn, current)
      }
      return toReturn; }
```

## Résultat

```
List<String> confs = Arrays.asList( "jug", "devoxx", "javaone" );
reduce( (s1, s2) -> s1 + ", " + S2, confs );
// jug, devoxx, javaone
```

## Java 8

```
confs.stream().reduce((s1, s2) -> s1 + ", " + s2 ).get() )
```

# Combinator

**Définition**

*“C'est une fonction qui définie une nouvelle fonction à partir de ses arguments ou d'autre combinators”*

# Null Combinator

## Problème

```
List< String > confs2 = Arrays.asList( "jug", "devoxx", "javaone", null );
map( s -> s.toUpperCase(), confs2);
// Exception in thread "main" java.lang.NullPointerException
```

## Solution

```
Function< String, String > nullCheck( Function< String, String > f ) {
      return (String s) -> s == null ? "null" : f.apply(s);
}
```

## Résultat

```
map( nullCheck(s -> s.toUpperCase()), confs2)
// [JUG, DEVOXX, JAVAONE, null]
```

# Composition

**Définition**

*“Combine plusieurs fonctions pour créer une nouvelle fonction”*

## Code

```
Function<String, String> compose (
                            Function<String, String> f1, Function<String, String> f2 ){
      return (String s) -> f1.apply( f2.apply(s) );
}
```

## Résultat

```
Function<String, String> up = (String s) -> s.toUpperCase();
Function<String, String> hello = (String s) -> "hello " + s;
up.apply( hello.apply("Erouan") );
compose( up, hello).apply("Erouan") ;
// HELLO EROUAN
```

## Java 8

```
hello.andThen(up).apply("Erouan")
```

***Twitter:*** *@ychartois*
***Github:*** *https://github.com/ychartois/ProgFoncJava8*