# Let the Types Work for You

Klarna Konferense

Felix Mulder

August 2018

## Agenda

- Functional Programming

## Agenda

- Functional Programming
- Type systems

## Agenda

- Functional Programming
- Type systems
- FP + Types == amazing!

## Agenda

- Functional Programming
- Type systems
- FP + Types == amazing!
- Profit!

## Bio

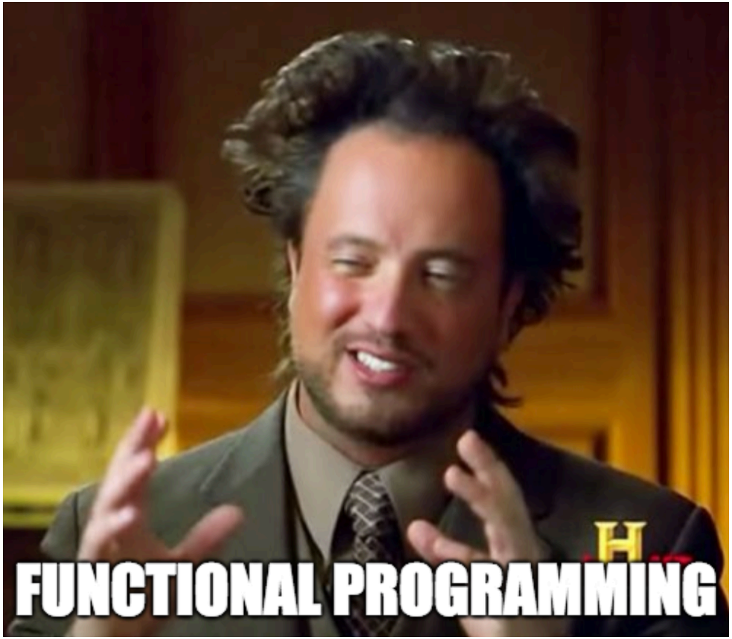
- Felix Mulder

## Bio

- Felix Mulder
- Software Engineer, Core Banking

## Bio

- Felix Mulder
- Software Engineer, Core Banking
- Compiler Engineer, Scala 3 @ EPFL


H
FUNCTIONAL PROGRAMMING

*“Do you know that feeling of having to hold too many things in your head at once?”*

## Functional Programming gets rid of that by definition.

## Referential Transparency

- Equational reasoning

```
x = 5
y = x + x
z = y + x

// ⟹
z = (5 + 5) + 5
```

- Compositionality

Referential Transparency + Types

==

Refactor All The Things! (without fear)

## Game over, OO. Right?

## What about the downsides?

## What if you could negate those downsides?

- Smarter inference

## What if you could negate those downsides?

- Smarter inference
- Better compiler messages

## What if we used the types to derive the implementation?

SAFELY ENDANGERED
WEBTOON
Who likes programming with types here?
oh geez ME ME ME!!!!
okay, let's learn how to use our type information to derive functions!

## Today we’re exploring type-level induction and recursion

## What we’re actually doing

Writing a compile-time serializer for data types - with no need for scary runtime reflection.

## Coding time!

## Why are we so obsessed with parametricity?

## Felix’s Conjecture

*“By being able to do anything, we can assume nothing”*

*“The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise”*

*– Edsger W. Dijkstra*

## Constraints Liberate, and Liberties Constrain

## `Any ⇒ Unit`

## Constraints Liberate, and Liberties Constrain

```scala
def foo(i: Int): Int = ???
```

## Constraints Liberate, and Liberties Constrain

```scala
def foo[A](a: A): A= ???
```

## Constraints Liberate, and Liberties Constrain

```scala
def foo[A](a: A): A = a
```

## Constraints Liberate, and Liberties Constrain

```scala
def id[A](a: A): A = a
```

## In Closing

- Type level recursion for fun and profit!

## In Closing

- Type level recursion for fun and profit!
- Built a type-level, compile-time JSON serializer

## In Closing

- Type level recursion for fun and profit!
- Built a type-level, compile-time JSON serializer
- You shouldn’t work against the compiler, make it work for you!