JAVA FAKTURA

# Na OOP świat się nie kończy

## Wstęp do programowania funkcyjnego

S01E08

Artur Czajka

# Ja

- Senior Dev / Tech Lead @ IDEMIA
- Programista-poliglota
  - Scala, Java, Bash, JavaScript, Python, C++, Haskell
- ~12k reputacji na StackOverflow za JavaScript ;)
- Wykładowca w CodersLab, a wcześniej SDA
- Gracz

@tr00per87

https://www.linkedin.com/in/arturczajka/

# Paradygmaty

JAVA FAKTURA

- Programowanie imperatywne
  - proceduralne (Fortran, Pascal, C)
  - obiektowe (Smalltalk, Objective-C, C++, Java, C#)
- Programowanie deklaratywne
  - funkcyjne (LISP, Haskell, Scala, Clojure)
  - logiczne (Prolog)

# Historia

JAVA FAKTURA

1957 - Fortran

1958 - LISP

1977 - John Backus, twórca Fortrana, odbiera nagrodę Turinga, a w swoim wykładzie mówi o tym, że **aby zapanować nad rosnącą złożonością tworzonych programów powinniśmy przejść na paradygmat funkcyjny**.

Niestety było już za późno ;)

# Już korzystasz z paradygmatu funkcyjnego!

JAVA FAKTURA

Jeśli w swoim kodzie:

- korzystasz głównie z interfejsów zamiast klas abstrakcyjnych
- preferujesz kompozycje nad dziedziczenie
- używasz typów generycznych
- używasz niezmiennych typów danych
- nie boisz się Streamów, Optionali i CompletableFuture'ów

To jesteś już skażony programowaniem funkcyjnym ;)

# Więcej!

JAVA FAKTURA

- Czyste funkcje
- Algebraiczne typy danych
- Funkcje wyższego rzędu
- Kompozycja funkcji
- Currying (Schönfinkeling)
- Typy wyższego rzędu
- Kontrola efektów ubocznych

# Czy programowanie funkcyjne mogło zapobiec wybuchowi w Czarnobylu?

https://www.digitaltrends.com/movies/chernobyl-hbo-miniseries-burning-questions/

# Wzorce projektowe

JAVA FAKTURA

W programowaniu obiektowym

1. Fabryka
2. Builder
3. Adapter/Dekorator/Proxy/Fasada
4. Singleton
5. Iterable
6. Visitor
7. Strategy

W programowaniu funkcyjnym

1. Funkcja
2. Funkcja (z curryingiem)
3. Funkcja (wyższego rzędu)
4. LOL. Nie.
5. Traversable
6. Funktor
7. Funkcja (wyższego rzędu)

# Funkcyjne wzorce projektowe

JAVA FAKTURA

**Funktor** - struktura danych dla której można zaaplikować funkcję na danych przechowywanych przez dany pojemnik

*id :: X -> X*
F.map(id) === F
F.map(A -> B).map(B -> C) === F.map(A -> B andThen B -> C)

**Funktor aplikatywny** - ulepszenie Funktora, możemy bezpośrednio aplikować funkcje, które znajdują się w pojemniku na elementach, które znajdują się w pojemnikach tego samego typu

**Monada** - kompozycja i aplikacja funkcji z efektami ubocznymi

*f :: A -> M<B>, g :: B -> M<C>*
new M(a).flatMap(f) === f(a)
new M(x).flatMap(M::new) === new M(x)
new M(a).flatMap(a -> f(a).flatMap(g)) === (new M(a).flatMap(a -> f(a)).flatMap(g)

**Monoid** - struktura ze zdefiniowaną łączną dwuargumentową operacją wewnętrzną (wynikiem działania jest struktura tego samego typu); operacja posiada przemienny element neutralny

**Traversable** - struktura, którą można przeglądać lub redukować element po elemencie

Rzućcie okiem na: http://adit.io/posts/2013-04-17-functors,_applicatives,_and_monads_in_pictures.html

# Przykłady monad w Javie

JAVA FAKTURA

**Optional** - wynik może być, ale może go nie być

**Stream** - niedeterministyczność operacji - nie wiadomo jak dużo będzie wyników

**CompletableFuture** - nie wiadomo, kiedy przyjdzie wynik, a operacja może się nie powieść

# Brakujące monady w Javie

JAVA FAKTURA

**Either** - wynik może być, ale może go nie być, jednak tym razem możemy odczytać, co poszło źle

**Try** - operacja może się powieść albo zakończyć wyjątkiem

**Validator** - operacja może się powieść albo nie, ale jeśli jest kilka takich operacji w sekwencji - możemy skumulować listę błędów

**Reader** - wykonywane działania mają dostęp do pewnych wartości tylko-do-odczytu (np. konfiguracja)

**Writer** - wykonywanie działań produkuje dodatkowe informacje, które można ze sobą łączyć (np. log)

# Zasady programowania ~~obiektowego~~

JAVA FAKTURA

W programowaniu obiektowym

1. **S**ingle Responsibility
2. **O**pen/Close
3. **L**iskov substitution
4. **I**nterface segregation
5. **D**ependency inversion

W programowaniu funkcyjnym

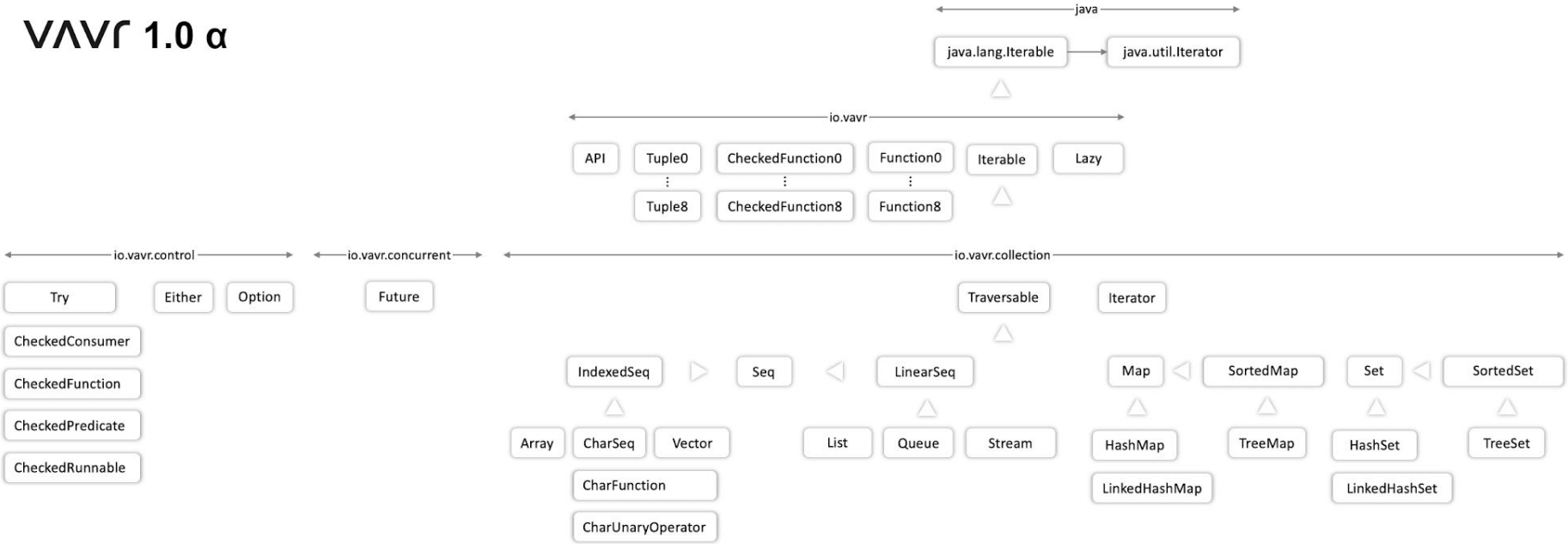
1. Czyste funkcje
2. Klasy typów (Type classes)
3. Prawa
4. Klasy typów
5. Funkcje wyższego rzędu i klasy typów ;)

# Lepsze kolekcje i...

JAVA FAKTURA

VAVR.IO

VAVR 1.0 α

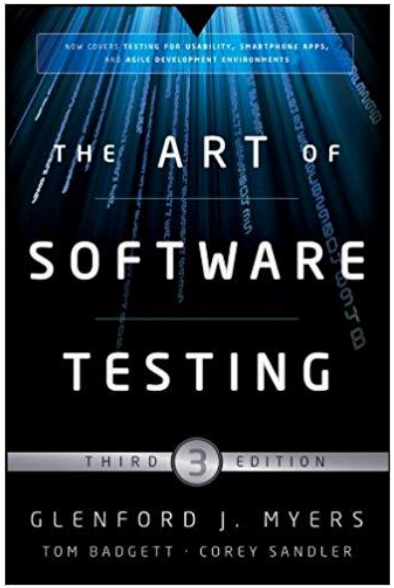



http://blog.vavr.io/purifying-the-api-surface-area/

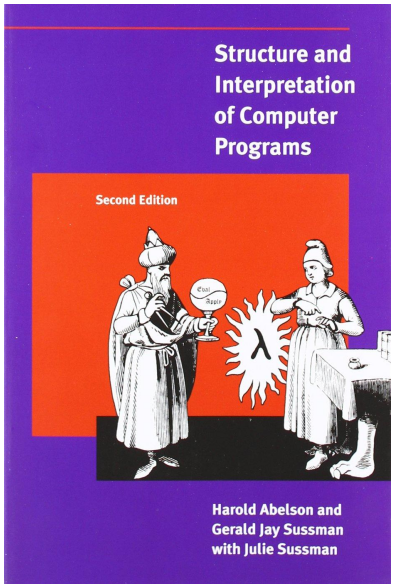
# Książki

JAVA FAKTURA

The Art of Software Testing



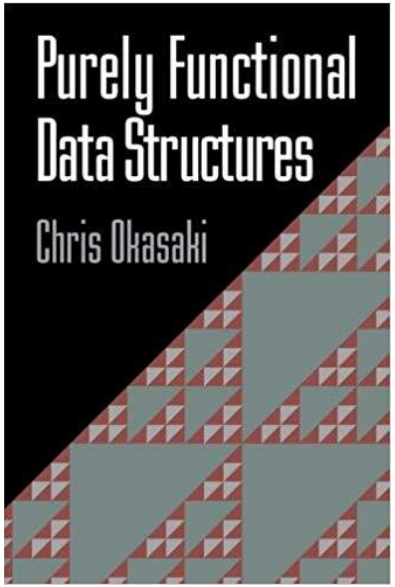

Structure and Interpretation of Computer Programs



# Książki

JAVA FAKTURA

Purely Functional Data Structures




Pearls of Functional Algorithm Design