Les types en SCALA

Volet 1 : généralités

Type en Scala : à quoi ça sert ?

Exemple:

https://elm-lang.org/

- Résoudre les erreurs potentielle dès la compilation
  - Structurer le code, les données
  - Avoir un code lisible, évolutif

Le typage

- SCALA est un langage typé

  (un type ~ un ensemble de valeurs)

- Est-ce suffisant pour caractériser Scala ?

- Quid des autres langages ?

Sortir de la bulle Scala et voir d’autres langages plutôt récents...

Langage non typé ?

- Wikipedia : « untyped language means the language does not have any type declaration. »

- L’unique : L’assembleur !

  Et le λ-calcul non typé...

Langages avec typage statique

- Langages avec typage statique :
  - Haskell, Scala, Idris…

    => type identifié à la compilation
    - Type explicite
    - Type implicite (inférence de type)
      - Left biased vs Right biased

Python

- Typage à la volée

```
a = "10"
a=1
print(a)

1
```

=> Typage dynamique

=> le type peut changer dans le temps ???

Langage avec typage dynamique

- Langages avec typage dynamique :
  - Clojure, Julia, Python

  => peu de points commun entre ces langages

- Dynamique => Kezako ?

Le typage dans Julia

- Stack overflow

A lot of blogs, and the manual itself, say that Julia is dynamically typed. But from my reading of the manual, it sounds to me more like it is statically typed with type inference, like F#.

- **Is Julia statically typed with type inference?**
- **Is it dynamically typed?**
- I'm assuming it is dynamically typed, it seems unlikely the manual is wrong.
- **Is type inference involved in Julia at all?**

Le typage dans Julia

https://docs.julialang.org/en/v1/manual/types/

« Julia's type system is dynamic, but gains some of the advantages of static type systems by making it possible to indicate that certain values are of specific types. »

=> Le type est résolu à la compilation si précisé, sinon au runtime ?

Compilation vs ...

- Compilé :
  - C++
- Interprété :
  - Python
- Autre ?
  - Pseudo-code + JIT sur la JVM
  - Est-ce tout ?

Le typage dans Julia

- Documentation Julia:

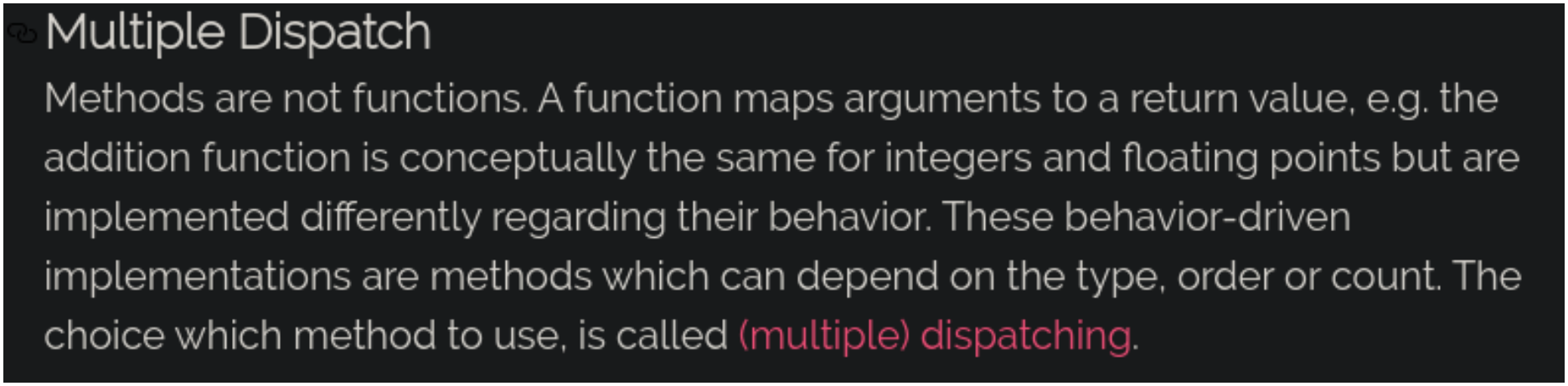
Quoting from the official Julia documentation:

> *Julia's type system is dynamic but gains some of the advantages of static type systems by making it possible to indicate that certain values are of specific types. This can be of great assistance in generating efficient code, but even more significantly, it allows method dispatch on the types of function arguments to be deeply integrated with the language. (More on* `https://docs.julialang.org/en/stable/manual/types/`*.)*

- Démo : julia-example.jl

Le typage dans Julia

- Remarque : multiple dispatch



http://cleverlytitled.blogspot.com/2010/01/dynamic-dispatch-in-scala.html

Julia compilation vs ...

- Julia fonctionne avec le compilateur LLVM JIT
  - Première exécution :
    - parsing + inférence de type (sinon Any)
    - Génération de code par le compilateur JIT
  - Seconde exécution :
    - Le code natif a déjà été généré

      => Exécution du code natif
- Démo : julia-example2.jl

Le typage dans Clojure

- Clojure est un langage fonctionnel
  - => **immutable**
  - => une variable n’est affectée qu’une seule fois !
  - => pourquoi alors parle-t-on de typage dynamique ?
- Dynamique ~ au moment de l’évaluation
  - => types identifiés au runtime
  - => interpréteur ?

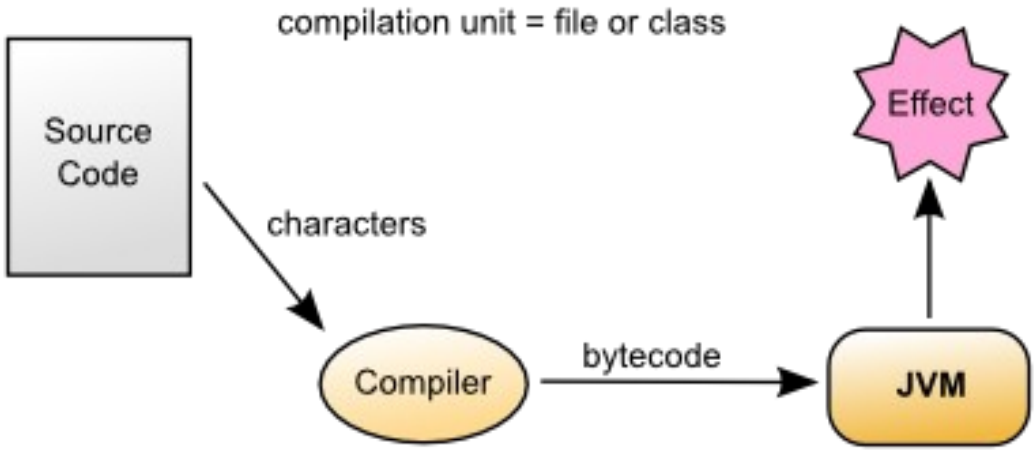
Le typage dans Clojure

- Stack overflow => « Clojure is strictly a compiled language, just like Java and Scala. The difference with Java is that it can also compile code at run-time and load it, instead of relying on a distinct compilation phase »

https://blog.ndk.io/clojure-compilation.html

Evaluation Clojure vs Java

- In Java, source code (.java files) are read as characters by the compiler (javac), which produces bytecode (.class files) which can be loaded by the JVM.



https://metebalci.com/blog/demystifying-the-jvm-interpretation-jit-and-aot-compilation/

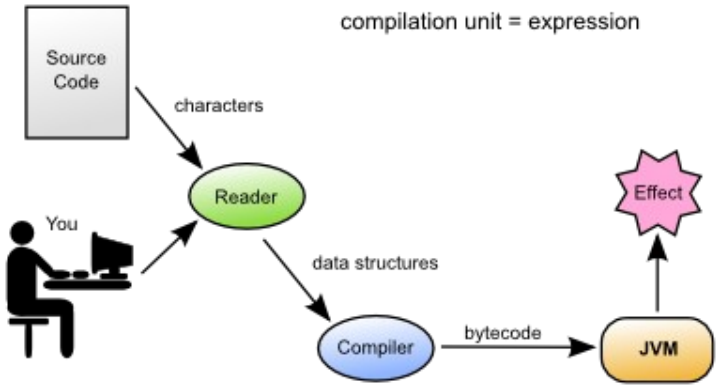
Evaluation Clojure vs Java

- Java
  - Interpréteur
  - JIT compiler
  - AOT compiler

https://www.baeldung.com/ahead-of-time-compilation

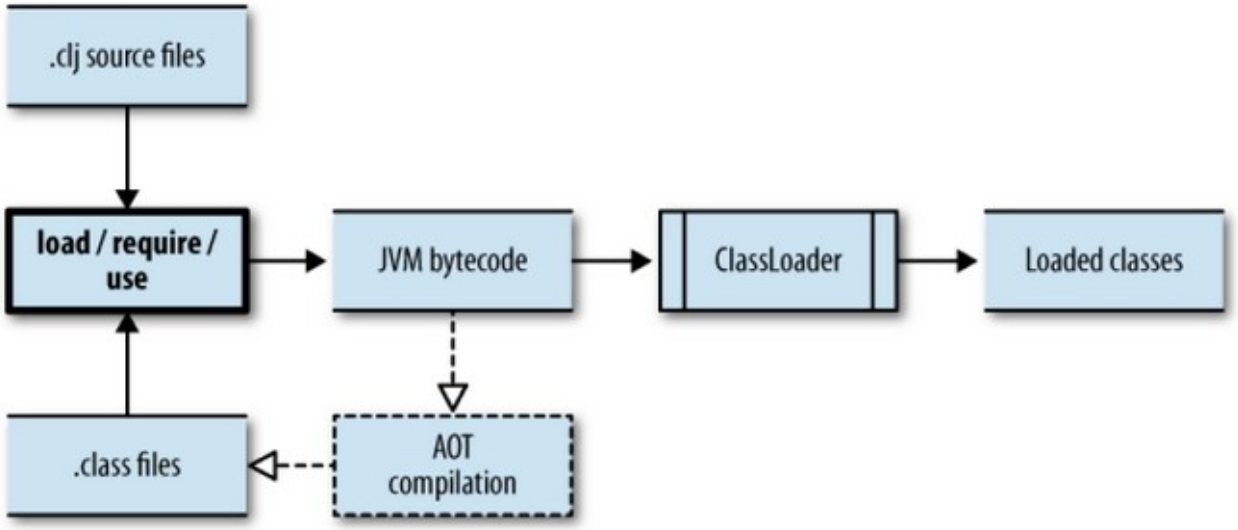
Evaluation Clojure vs Java

- In Clojure :
  - Reader lit des caractères (code source).
  - Source : fichier .clj ou expressions REPL
  - Reader produit des data Clojure.
  - Le compilateur Clojure : bytecode pour la JVM



Le typage dans Clojure

- Clojure toujours compilé
  - Au runtime : REPL ou chargement d'un .clj depuis le disque => compilation bytecode + chargement dans JVM. Bytecode + class => effacé à la fin de l'exécution
  - AOT compilation : le résultat est sauvegardé sur disque



- En clojure, on peut compiler du code source pendant l’exécution

  => langage « Homoiconique » (cf. Julia, prolog)

  ~ méta programmation

  ~ macro (data => data)

Aspect dynamique ~ interactivité

Typage en Clojure

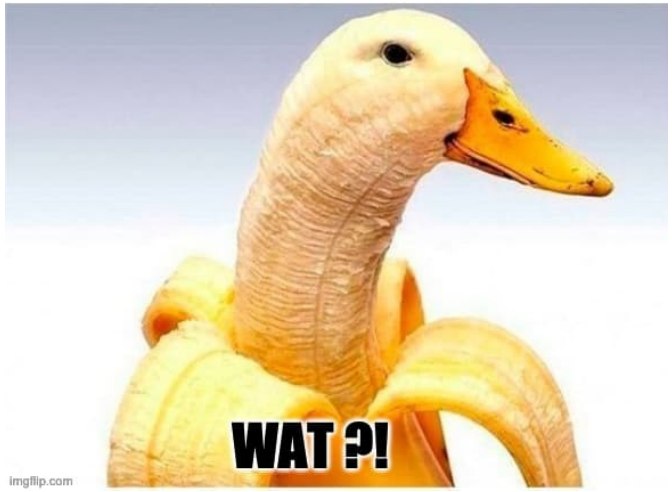
- Pb : changer T en Option[T] partout en scala par exemple
- programmation par contrat (granularité > typage)
- Specs
  - Démo : clojure-specs

Typage en Clojure

- Polymorphisme
  - multiple dispatch (≠ java/scala)
  - Predicate-dispatch
    - démo : clojure-predicate-dispatch
  - pattern matching
  - protocols, datatypes
- Extension code source for free (cf. typeclasse)

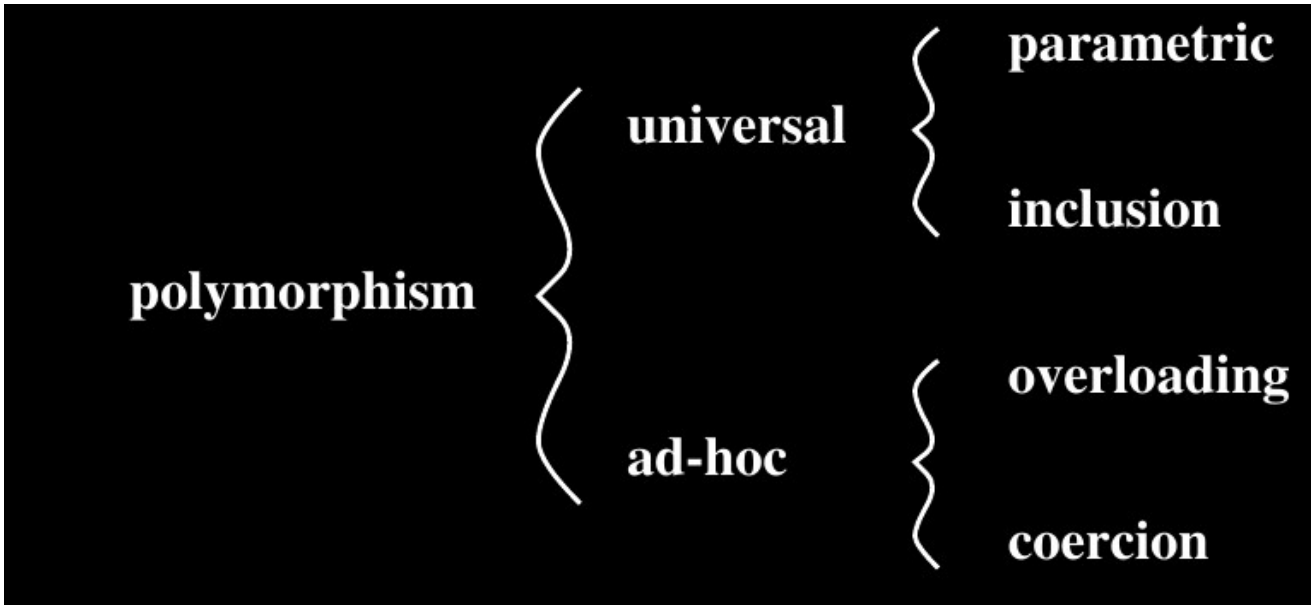
Polymorphisme

- Héritage
- Duck typing (démo)



Polymorphisme

- Classification



http://lucacardelli.name/Papers/OnUnderstanding.pdf

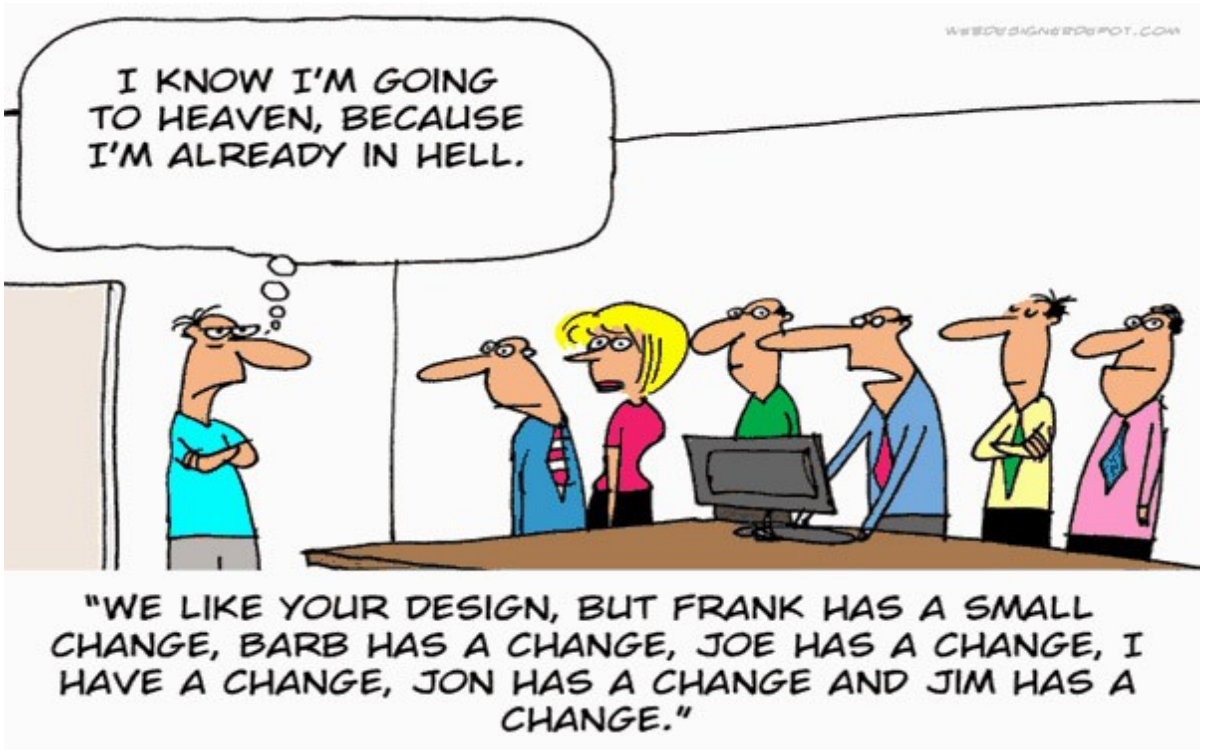
Valeurs et variables peuvent avoir plusieurs types

Polymorphisme

- Row polymorphism
  - Structural typing
- Parametric polymorphism
  - cf. Identity
- Différences polymorphism / inheritance :
  - Polymorphisme : appliqué aux function
    - Compile-time (overloading)
    - Run-time (overriding)
  - Héritage : appliqué aux classes

Conclusion

- Langages récents :
  - Compilé statiquement ou au runtime 1 fois
  - Méta-programmation
- Typage vs non typage
  - Programmation par contrat + règles
  - Inférence de type + compilation la 1ère fois
- Scala
  - Pas ces avantages ci-dessus => quels sont les avantages du typage statique ?


I KNOW I'M GOING TO HEAVEN, BECAUSE I'M ALREADY IN HELL.
"WE LIKE YOUR DESIGN, BUT FRANK HAS A SMALL CHANGE, BARB HAS A CHANGE, JOE HAS A CHANGE, I HAVE A CHANGE, JON HAS A CHANGE AND JIM HAS A CHANGE."