1 DÉFINITION DE TYPES

Types personnalisés

On crée un type avec le mot-clé `data`. Au plus simple, un type réunit plusieurs variables membres :

```
ghci> data MonType = MonType String Integer
ghci> let a = MonType "abc" 123
```

(La première occurence de MonType est le nom du type, la seconde est le nom du constructeur)

■ **Accesseurs explicites (*record syntax*)** Un accesseur explicite définit automatiquement une fonction qui prend un objet du nouveau type en paramètre et renvoie la valeur

```
data Book = Book {
    bookISBN  :: Int,          -- bookISBN :: Book -> Int
    bookAuthors  :: [String], -- bookAuthors :: Book -> [String]
    bookTitle  :: String       -- bookTitle :: Book -> String
    }
```

■ **Types algébriques** Un type algébrique présente une alternative, en offrant plusieurs constructeurs. Le plus répandu est `Maybe` :

```
data Maybe a = Nothing | Just a
```

Tous les types définis avec `data` sont algébriques, même s'ils n'offrent qu'un seul constructeur (O'Sullivan et al. 2008).

■ **Types récursifs** Le type liste natif est défini récursivement. On peut le réimplémenter comme suit :

```
data List a = Empty | Cons a (MyList a)
```

Synonymes de types

`type` crée un synonyme d'un type existant. Les synonymes et le type auquel ils renvoient sont interchangeables.

```
type ObjectId = Int16
```

Les synonymes créés avec `type` peuvent servir :

- À clarifier le sens des champs dans les types personnalisés sans accesseurs (`type ISBN = Int` pour un type Book, par exemple).
- Comme notation abrégée pour des types complexes fréquemment utilisés.

```
type Authors = [String]
type Title = String
type ISBN = Int
type Year = Int
data Book2 = Authors Title Year ISBN
```

Synonymes « forts »

- **Maybe**
- **Either**

2 CLASSES DE TYPE (*TYPE CLASSES*)

Les classes de type ne sont pas des classes au sens que ce terme possède en POO. Elles sont plus proches de ce qu'on nomme des interfaces : elles décrivent des fonctions pour lesquelles un type qui appartient à la classe fournit une implémentation.

```
1 class Boolable a where
2     toBool  :: a -> Bool
```

Références

O'Sullivan, Bryan, Don Stewart et John Goerzen (2008). *Real World Haskell*. Anglais. O'Reilly. URL : http://book.realworldhaskell.org/.