$\times$ `List`, mais comme l'union disjointe du singleton `{null}` et de ce produit cartésien. Un tel type s'appelle un type *disjonctif*.

Un autre exemple est celui des *expressions arithmétiques*. Une expression arithmétique est ou bien une constante entière, par exemple `3`, ou bien une variable, par exemple `x`, ou bien la somme de deux expressions arithmétiques, par exemple `y + 3` ou `x + (y + 3)`, ou bien le produit de deux expressions arithmétiques, par exemple `x` $\times$ `(y + 3)`.

De ce fait, nous voudrions définir le type `Expr` par

`Expr = int` $\uplus$ `String` $\uplus$ `(Expr` $\times$ `Expr)` $\uplus$ `(Expr` $\times$ `Expr)`

Quand un type peut se définir, éventuellement récursivement, comme l'union disjointe d'un singleton et d'un produit cartésien, on identifie simplement l'élément du singleton avec la valeur `null`. Cette méthode marche par exemple pour les listes, ou les arbres binaires `Arbre = {feuille}` $\uplus$ `(Arbre` $\times$ `Arbre)` mais pas pour le type des expressions arithmétiques.

Une autre solution est de plonger ce type dans un type plus vaste qui contient un champ appelé *champ sélecteur* que l'on peut, par exemple, noter `select` et qui indique le cas dans lequel on se trouve : constante, variable, somme ou produit. Puis on met suffisamment de champs pour couvrir tous les cas. Dans l'exemple des expressions arithmétiques, il faut un champ de type `int` qui ne sert que dans le cas où l'expression est une constante, un champ de type `String` qui ne sert que dans le cas où l'expression est une variable, et deux champs de type `Expr` qui servent dans le cas où l'expression est une somme ou un produit. Cela mène à la définition suivante.

```
class Expr {
 int select;
 int val;
 String var;
 Expr arg1;
 Expr arg2;}
```

Quand la valeur du champ sélecteur est `0`, c'est-à-dire que l'expression est une constante, on remplit les champs superflus, `var`, `arg1` et `arg2` par des valeurs quelconques, par exemple, `""`, `null` et `null`. Notons que, pour que de telles valeurs par défaut existent, il faut que les types des champs soient non vides, en particulier que le type `Expr` lui-même soit non vide. En Java, tous les types enregistrements sont non vides car ils contiennent une valeur `null`.

Exercice 5.3

1. Écrire une fonction qui prend en argument une expression arithmétique et l'affiche en notation mathématique traditionnelle, avec des parenthèses.