la fonction `assoc`.

Chacune de ces solutions a des inconvénients : la première obscurcit les programmes, puisqu'elle utilise, pour exprimer l'absence de valeur, une valeur conventionnelle dont la signification n'est pas immédiate. La deuxième et la troisième alourdissent le type de la fonction `assoc`. Et la dernière demande de parcourir deux fois la liste d'associations.

7.2 Les exceptions

Une alternative est d'utiliser une construction appelée *exception*. On écrit alors la fonction `assoc` ainsi

```
static int assoc (final int x, final List l) throws Exception {
 if (l == null) throw new Exception();
 if (x == l.key) return l.val;
 return assoc(x,l.tl);}
```

Cette fonction est une fonction partielle qui renvoie la valeur associée à la clé `x` dans la liste `l` quand cette valeur existe et qui échoue sinon. On dit dans ce cas que la fonction *lève* une exception. L'instruction pour lever une exception est `throw e ;` où `e` est une valeur de type `Exception`. La manière la plus simple de fabriquer une valeur de type `Exception` est d'utiliser la construction `new` : `new Exception() ;`.

Le fait que la fonction `assoc` puisse lever une exception doit se déclarer en insérant `throws Exception` entre la liste des arguments et le corps de la fonction. Il faut prendre garde à ne pas confondre les mots clés `throw`, deuxième personne du pluriel de l'impératif présent, qui est utilisé pour lever une exception, et `throws` troisième personne du singulier de l'indicatif présent, qui est utilisé pour indiquer qu'une fonction peut lever une exception.

7.3 Récupérer les exceptions

Quand on appelle une fonction qui peut ainsi lever une exception, on peut indiquer quoi faire dans le cas où cette fonction renvoie cette exception. Cela se fait avec la construction `try p catch (Exception e) q ;`, par exemple

```
try {System.out.println(assoc(x,l));}
catch (Exception e) {System.out.println("Pas dans la liste");}
```