Cette possibilité de laisser les exceptions se propager de fonction en fonction est le principal avantage du mécanisme des exceptions sur le mécanisme qui consisterait à renvoyer un booléen en plus de la valeur associée à la clé `x`. Dans ce cas, la fonction `assocplus` devrait s'écrire, comme ci-dessus, de manière beaucoup plus lourde

```
static Pair assocplus (final int x, final List l) {
 Pair p = assoc(x,l);
 if (p.b) return new Pair(true,p.v + 1);
 return p;}
```

Ceux qui aiment les métaphores pourront comparer le comportement de la fonction `assocplus` quand la clé n'est pas dans la liste à celui d'un étudiant quand la sirène d'incendie résonne au milieu d'un TP de chimie. Il a été dit une fois que dans ce cas, il faut sortir du laboratoire et se rassembler à quelques mètres du bâtiment. Mais la feuille de TP prescrit simplement de mélanger deux produits A et B dans un tube à essai, puis de chauffer le mélange et de rajouter un troisième produit C, sans mentionner la sirène. Et cette feuille de TP serait beaucoup plus difficile à lire si elle avait la forme "Si la sirène d'incendie résonne, alors sortir du bâtiment, sinon mélanger les produits A et B, puis si la sirène d'incendie résonne, alors sortir du bâtiment, sinon chauffer le mélange, puis si la sirène d'incendie résonne, alors ..."

7.5 Les messages d'erreur

Certaines primitives de Java peuvent déclencher des exceptions. Ainsi, l'instruction

```
System.out.println(1/0);
```

déclenche l'exception

```
java.lang.ArithmeticException: / by zero
```

Les *messages d'erreur* de Java sont donc simplement un cas particulier des exceptions.

7.6 La sémantique des exceptions

Définir la fonction Σ dans un langage qui comporte des exceptions demande un certain travail. Nous avons vu que le résultat de l'exécution d'une instruction