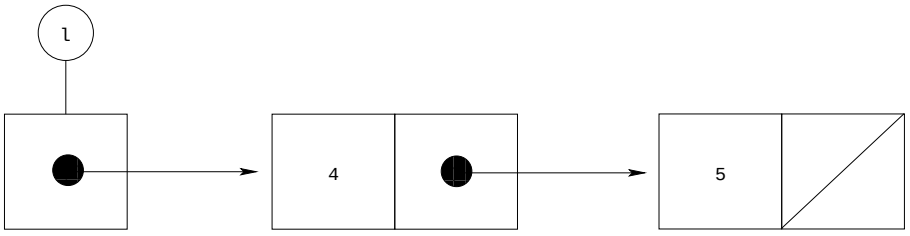

On peut aussi définir un constructeur dans la déclaration du type `List`

```
List (final int x, final List y) {this.hd = x; this.tl = y;}
```

et écrire ce programme de manière plus concise

```
List l = new List(4,new List(5,null));
```

Il semble donc bien que cette définition récursive définisse un type dont les éléments sont les listes d'entiers.

Exercice 5.1

Écrire une fonction qui calcule la somme des éléments d'une liste d'entiers.

5.1.4 Les définitions récursives et les équations au point fixe

Au paragraphe précédent nous avons « défini » le type `List`, comme `List = {null} ⊎ (int × List)`. Le fait que `List` apparaisse à droite du signe `=` est dû au fait que la définition est récursive. Encore une fois, les définitions récursives nous apparaissent comme des définitions circulaires et une manière d'éviter cette circularité est de lire la proposition `List = {null} ⊎ (int × List)`, non comme une définition, mais comme une équation. Le type `List` est donc défini comme la solution de l'équation `List = {null} ⊎ (int × List)`. Encore une fois, se pose la question de l'existence et de l'unicité de cette solution.

Pour construire une solution on procède, comme pour le cas des fonctions récursives, par approximations successives, en définissant l'ensemble L_i des valeurs de type `List` que l'on peut construire en `i` étapes au plus.

$L_0 = \varnothing$
$L_1 = \{null\} \uplus (int \times L_0) = \{null\}$
$L_2 = \{null\} \uplus (int \times L_1)$
$L_3 = \{null\} \uplus (int \times L_2)$
...