Exercice 1

Les exercices qui suivent portent tous sur les ensembles d'entiers vus au cours (classe `IntSet` et ses sous-classes). Le but de ce premier exercice est d'ajouter à ces classes une fonction permettant d'afficher le contenu d'un ensemble, afin de faciliter les tests.

Toutefois, nous n'allons pas définir directement une fonction d'affichage. Comme d'habitude, nous allons commencer par définir une fonction plus générale que nous spécialiserons ensuite pour l'affichage.

Cette fonction plus générale se nomme `foreach`. Elle prend en argument une fonction et applique cette fonction à tous les éléments de l'ensemble. Elle a le profil suivant :

```
def foreach(f: int => unit): unit
```

La fonction `foreach`, ainsi que la fonction `f` qu'elle prend en argument, est déclarée comme retournant une valeur du type `unit`. Il n'existe qu'une valeur de ce type, qui s'écrit `()` et qui se prononce également *unit*. On utilise ce type lorsqu'on ne s'intéresse pas à la valeur de retour d'une fonction. [1]

Ainsi, au moyen de cette nouvelle fonction, pour afficher tous les éléments d'un ensemble `s`, on écrit simplement :

```
s.foreach(java.lang.System.out.println)
```

Exercice 2

Ajoutez une fonction `intersect` aux ensembles d'entiers, qui calcule l'intersection de deux ensembles.

Conseil : commencez par ajouter une fonction auxiliaire `intersect0` qui prend un ensemble accumulateur comme second argument. Cet accumulateur contient le résultat courant de l'intersection. Il joue un rôle similaire à celui de l'argument `result` de la fonction `iter` utilisée dans la définition de `sum` (voir cours, p. 18, semaine 2).

```
def intersect(that: IntSet): IntSet;
def intersect0(that: IntSet, accu: IntSet): IntSet;
```

La définition de `intersect` en termes de `intersect0` est ensuite triviale.

[1] En Java, on utiliserait pour cela le type `void`. Malheureusement, le type `void` ne contient aucune valeur, ce qui ne convient pas en Scala où toutes les expressions doivent avoir une valeur.

Exercice 3

Ajoutez une fonction `filter` aux ensembles d'entiers, qui filtre un ensemble au moyen d'un prédicat. `filter` prend en argument une fonction, le prédicat, qui reçoit un entier et qui retourne un booléen. `filter` retourne ensuite le sous-ensemble de tous les entiers de l'ensemble d'origine pour lesquels le prédicat est vrai. Par exemple, l'appel suivant :

```
s.filter(x => x > 0)
```

appliqué à l'ensemble $\{-10, 5, 21, -1, 0, 3\}$ retourne l'ensemble $\{5, 21, 3\}$, c'est-à-dire le sous-ensemble des positifs.

Conseil : utilisez une technique similaire à celle utilisée pour l'exercice précédent.

Exercice 4

Écrivez une nouvelle version de la fonction `intersect` au moyen de la fonction `filter` définie à l'exercice prédécent.