Les fonctions ne sont pas à écrire entièrement, faites simplement un plan. Il est conseillé d'implémenter ces classes chez vous.

On considère l'interface suivante.

```
interface Evaluable {
      int toInt();
        // renvoie un entier qui evalue l'instance
      String toString();
        // renvoie une String qui donne la facon se s'afficher de l'objet
}
```

Exercice 1 Concevoir une classe `Date` qui implémente `Evaluable` ; elle s'affiche comme `Dimanche 27 septembre 2015` et s'évalue au nombre de jours écoulés depuis le 1er janvier de l'an 1. Elle doit être capable d'incrémenter d'un jour, de vérifier si elle est cohérente avec une autre date : par exemple la date précédente n'est pas cohérente avec `Jeudi 28 Septembre 2015`. (On ignore les années bissextiles.)

Exercice 2 Concevoir une classe `ListeTriee` sur le modèle de `PileExecutable` dont les chaînons doivent implémenter `Evaluable`. Elle doit être capable d'afficher ses éléments par deux méthodes différentes, l'une appelant `toString()`, l'autre `toInt()`. Quand on ajoute un élément, il doit être placé *au bon endroit* en fonction de son évaluation : les éléments de la liste doivent toujours être triés par `toInt()` croissant.

Comment doit-on modifier `Date` pour pouvoir l'utiliser avec cette chaîne ?

Peut-on modifier ListeTriee pour pouvoir utiliser `Date` telle quel ?

On considère maintenant le code suivant

```
interface Fonction {
    int appliquer(int x);
}

class FonctionMap extends ArrayList<Fonction> {
  public int appliquer (int x) {
    int y = x;
    for (Fonction f: this)
      y = f.appliquer(y);
    return y;
  }
}

class Carre implements Fonction {
  int appliquer (int x) { return (x*x); }
}
class PlusTrois implements Fonction {
  int appliquer (int x) { return (x+3); }
}
```

Exercice 3 Que fait la classe `FonctionMap` ? Qu'affiche le code suivant ?

```
FonctionMap f = new FonctionMap();
f.add(new PlusTrois());
System.out.println(f.appliquer(1));
f.add(new Carre());
System.out.println(f.appliquer(1));
```

Comment ajouter facilement des `Fonction` à `f` ? Ajouter à `f` une fonction qui multiple par 2 à l'aide d'une classe locale. Ajouter à `f` une fonction qui soustrait 4 à l'aide d'une classe anonyme. Qu'affiche alors `System.out.println(f.appliquer(1))` ;

Exercice 4 On considère le code suivant.

```
Class Poly extends FontionMap {
  enum Atom implements Fonction {
    PlusOne, TimesX;

    static int xvalue = 1;
    public int appliquer(int x) {
      if (this == PlusOne)
        return (x+1);
      else
        return x*xvalue;
    }
  }
  public int appliquer(int i) {
    Atom.xvalue=i;
    return super.appliquer(0);
  }
}
```

Où peut-être placé le code suivant ? dans ce cas, qu'affiche-t-il ?

```
System.out.println(Atom.PlusOne.appliquer(1))
```

Qu'affiche le code suivant ?

```
FonctionMap g = new FonctionMap();
g.add( Atom.PlusOne );
g.add( Atom.TimesX );
g.add( Atom.PlusOne );
g.add( Atom.PlusOne );
g.add( Atom.TimesX );
System.out.println(g.appliquer(0));
System.out.println(g.appliquer(3));
```

Qu'est en réalité la fonction `g.appliquer(i)`. Quelle classe de fonctions peut-on réaliser en ajoutant uniquement `Atom.PlusOne` et `Atom.TimesX` ? Écrire le code pour la fonction $i \mapsto (i^2+1)^2$.

Exercice 5 L'élève avisé aura remarqué que `FonctionMap` satisfait elle-même l'interface `Fonction` ; on considère dorénavant qu'elle a été déclarée comme suit.

```
class FonctionMap extends ArrayList<Fonction> implements Fonction
```

Le code suivant est-il correct et si oui, qu'affiche-t-il ?

```
f.add(g);
System.out.println(f.appliquer(0));
f.add(f);
System.out.println(f.appliquer(0));
```