**TD 1 : Programmation fonctionnelle – Généricité**

**Exercice 1 : Propriétés fonctionnelles**

**Public String To lowercase( )** : méthode qui prend une chaine et renvoie un String. Paramètre implicite qui reçoit l’objet en fonctionnel. Cela ne serait pas fonctionnel si cela modifie la chaine sur place. Il y a aussi un paramètre implicite qui utilise la localisation de notre appareil, selon la langue.

Ce n’est pas fonctionnel car si on change de pays, on n’aura pas le même résultat. Du point de vu local c’est fonctionnel car elle appelle une fonction locale.

**Max** n’a pas d’effet random, elle semble fonctionnelle.

**Sin** : est ce que not ou number est un null. Si not ou number est une valeur comme les autres alors c’est fonctionnel.

**List ::add**-> Cela rajoute un élément à la fin de la liste. Cela retourne vrai lorsque cela réussi et cela peut soulever des exceptions. Ce n’est pas fonctionnelle car une fonction n’est pas censée soulever une exception et elle fait un effet de bord, elle modifie une structure. Une méthode qui serait fonctionnelle, elle ne lèverait pas d’exception et ce serait un Add(T e) : List<T>.

**List :: size** ->

**File ::length** -> Elle lève une exception. Ce n’est pas fonctionnel. De plus, c’est une entrée-sortie. Cela ouvre le fichier donc il pourrait y avoir un problème si cela modifie le fichier.

C’est quoi l’idée de cet exercice ? Lorsqu’on programme, on fait des méthodes, et puis on se base sur des API de bas niveau pour définir d’autre API. On a des éléments centraux qui s’appuient sur des API et on rajoute du code par-dessus. Plus on est au cœur, plus on va être fonctionnel. On va essayer de se débarrasser du random et des entrées-sorties.

**M1** n’est pas fonctionnelle car on change a.

**M2** n’est pas fonctionnelle stricto sensu, si on maîtrise le changement de a, la méthode est fonctionnelle. Elle n’est pas fonctionnelle car elle dépend d’une fonction qu’on peut changer.

**M3** n’est pas fonctionnelle. Car on modifie un fichier.

**M4** est fonctionnelle. Cela ouvre la voie à la mémoïsation, cela se base sur le fait qu’une fonction retourne toujours le même résultat lorsque c’est la même entrée. On peut stocker le résultat quelque part. On va créer pas exemple une table de hachage.

**M5** n’est pas fonctionnelle car cela dépend d’une valeur externe qui peut être modifiée. Elle serait fonctionnelle si on a le contrôle sur la valeur. On doit remplacer a par c, si on veut qu’elle soit fonctionnelle. d est privé donc on peut aussi remplacer a par d.

M6 n’est pas fonctionnelle.

**M7** est fonctionnelle.

**M8** est fonctionnelle.

**M9** est fonctionnelle. Qu’est ce qui se passe si on me passe zéros.

**M11** est fonctionnelle car c’est un type de base dans le paramètre, ainsi l’entier est passé par valeur, il aura donc pas bougé. Si c’était une classe, alors l’objet a été passé par référence et l’objet peut être modifié. Il y a une exception : String est une classe mais les String sont stockés comme des constances.

# Exercice 3

N : Nœud { fs : [Arbre]}= la taille du nœud (n) est la somme de tous les size(f) avec f appartenant à fs.

La taille du nœud N

**int** size(){

**int** r=0;

    for(Arbre f: fs){

        r+= f.size();

    }

    return r;

}

Boolean continet(final Integer val);

Contient : Noeud\*N-> B