# Travaux Dirigés de Java n°3

— Etienne.Duris@univ-mlv.fr —

**▸ Exercice 1. (Liste récursive implantée par une classe)**

*Écrire une classe* `List` *représentant une liste récursive d'objets. Cette classe contient deux champs,* `head` *et* `tail` *ainsi qu'une constante* `NIL` *représentant une liste vide. Écrire une méthode* `int length()` *qui retourne la longueur de la liste.*

*Quels constructeurs définir pour cette classe ? Avec quelles visibilités ?*

*Redéfinir la méthode* `toString()` *pour cette classe* `List`*.*

**▸ Exercice 2. (Interface `List` et son implantation)**

*Définir une interface* `List` *avec sa méthode* `length()`*. Définir deux classes* `Nil` *et* `Cons` *implantant l'interface* `List` *et représentant les deux types de constituants d'une liste. Ces classes doivent naturellement implanter la méthode* `length()`*, mais également fournir chacune au moins un constructeur.*

*Que faire de la constante* `NIL` *? Que deviennent les visibilités des constructeurs ?*

*Ajouter à l'interface* `List` *une méthode* `List append(List l)` *permettant de concaténer de manière applicative (c'est-à-dire sans effets de bord) la liste* `l` *à la liste courante et de retourner la concaténation des deux listes.*

**▸ Exercice 3. (Minimum d'une liste)**

*La liste représentée par l'interface* `List` *permet de stocker tout type d'objets (*`Object`*) et on veut maintenant ajouter une méthode* `min()` *à cette interface qui retourne le "plus petit élément" de la liste. Pour implanter cette méthode, il faut être capable de comparer un élément de la liste à un autre. Cette propriété peut typiquement être représentée par une interface déclarant la méthode* `compareTo()` *que la classe de chaque objet stocké dans la liste devra implanter si l'on veut pouvoir utiliser la méthode* `min()`*.*

*Écrire cette interface. Modifier la classe* `Disque` *vue dans le TD précédent de sorte à ce qu'elle l'implante. Déclarer ensuite la méthode* `min()` *dans l'interface* `List` *et la définir dans les classes qui l'implantent.*

**▸ Exercice 4. (Expressions arithmétiques)**

*On désire manipuler des expressions arithmétiques sur des* `double` *et, en particulier, on souhaite pouvoir les évaluer.*

*Définir l'interface* `Evaluable` *et sa méthode* `double eval()` *que devront implanter les classes d'expressions arithmétiques.*

*Définir la hierarchie des classes d'expressions. On pourra considérer que :*

- *une expression générale est représentée par une classe abstraite* `Expr`*;*
- *un opérateur général est représenté par une classe abstraite* `Operator` *qui définit une priorité et une représentation sous la forme* `String` *de cet opérateur, ainsi qu'un constructeur permettant d'initialiser ces deux champs ;*
- *les opérateurs unaires (*`Unary`*) et binaires (*`Binary`*) sont des opérateurs particuliers, définissant respectivement comme champs une opérande ou bien deux. Pour chacun de ces deux types, on doit être capable de définir un constructeur initialisant le ou les champs ainsi que la méthode* `toString()`*;*
- *les opérateurs "concrets" habituels (*`Plus`*,* `Div`*,* `UMinus`*, etc...) héritent des opérateurs binaires ou unaires. La méthode* `eval()` *peut être implantée dans chacun d'eux, conformément à leur sémantique ;*
- *les constantes et les variables sont des expressions (*`Expr`*) particulières qui associent une valeur à un nom. Seules les variables peuvent être affectées à une nouvelle valeur après leur création. On pourra discuter des différentes hierarchies de classes permettant de représenter les constantes et les variables.*

*Créer différents objets de type* `Expr` *et tester les méthodes définies. On ne demande pas ici d'analyser une expression saisie au clavier (pas d'analyse lexicale ou parsing), mais simplement de la créer dans une méthode* `main()` *et de l'évaluer ou de l'afficher.*

*Que penser de l'affichage obtenu. Revoir la définition des méthodes* `toString()` *afin d'ajouter des parenthèses dans la représentation des expressions, mais seulement là ou c'est indispensable (en fonction des priorités des opérateurs).*

### ▸ Exercice 5. (Énumération d'une liste)

*Dans l'interface* `List` *de l'exercice précédent, on désire maintenant pouvoir obtenir un énumérateur (*`java.util.Enumeration`*) des éléments de la liste.*

*Pour cela, déclarer une méthode* `Enumeration elements()` *dans l'interface* `List` *et l'implanter dans les classes* `Nil` *et* `Cons`*. Dans chacune de ces classes, l'appel à* `elements()` *retourne une instance d'une classe interne statique qui implante l'interface* `Enumeration`*.*

*Que penser de votre hierarchie de classes et d'interfaces. Peut-on imaginer de transformer l'interface* `List` *en une classe abstraite ?*

*Remarque : depuis la version 1.2 de Java, il est recomandé d'utiliser plutôt* `java.util.Iterator` *que* `java.util.Enumeration`*.*

*Question subsidiaire : comment pourrait on obtenir une énumeration "dynamique", c'est-à-dire qui prenne en compte les ajouts éventuels à la liste pendant l'utilisation de l'énumeration ?*