Programmation Orientée Objet
L2S4 2007–2008
hoenen@icps.u-strasbg.fr

Université Louis Pasteur
UFR de math-info
1er avril 2008

# TD5 : classes abstraites et interfaces

## Exercice 1 : *interfaces*

1. Construisez une classe `Couple` qui possède :
   - deux attributs entiers,
   - une méthode `addition` qui prend en paramètre un autre couple et renvoie le couple égal à la somme des 2 couples,
   - une redéfinition de la méthode `toString` pour permettre de décrire l'objet.
2. Construisez une classe `Triplet` qui fait la même chose que `Couple` mais pour 3 entiers. `Triplet` doit hériter de `Couple` et doit réutiliser au maximum les méthodes de la classe `Couple`.
3. Définissez une interface `Nuplets` qui assure que la classe qui l'implémente possède deux méthodes `somme` et `produit` qui calculent respectivement la somme et le produit des attributs entiers d'une classe représentant un Nuplet.
4. Construisez les classes `CouplePlus` et `TripletPlus` qui héritent des classes des questions 1 et 2 et qui implémentent `Nuplet`.
5. Écrivez une classe *Test* qui contient une méthode `sommeProduit` qui prend en paramètre un Nuplet et renvoie un couple dont le premier élément est égal à la somme des éléments du Nuplet, et le second élément est égal au produit des éléments du Nuplet.
6. On veut maintenant que les Nuplets possèdent une méthode `plusGrand` qui permette de les ordonner selon la valeur de leur produit. On ne veut pas modifier l'interface `Nuplet` pour pouvoir conserver `CouplePlus` telle quelle. Trouvez une solution réutilisant les différentes classes et interfaces déjà implantées qui permette d'obtenir les classes `CoupleOrdre` et `TripletOrdre`.

## Exercice 2 : *classes abstraites*

1. Construisez une classe abstraite `ListeTri` qui correspond à une liste triée d'objets. Cette classe doit notamment contenir :
   - un attribut `liste` (par exemple du type `Vector`),

   et différentes méthodes qui peuvent être implantées ou abstraites :
   - une méthode `plusGrand` qui prend deux objets en paramètre et qui renvoie `true` si le premier est plus grand que le deuxième,
   - une méthode `ajouter` qui insère un objet dans la liste en respectant l'ordre croissant,
   - une methode `toString` qui renvoie une chaîne de caractères représentant la liste.
2. Construisez la classe `ListeTriCouple` qui hérite de `ListeTri` et utilise ordonne des objets de type `CouplePlus` de manière lexicographique.
3. Une solution était-elle envisageable uniquement avec des interfaces ? Quel est l'intérêt ici des classes abstraites ?

## Exercice 3 : *conception autour d'un Tuner*

Un `Tuner` est un récepteur d'ondes qui sait renvoyer sa fréquence actuelle, augmenter sa fréquence ou la diminuer selon un certain pas. Selon les longueurs d'ondes qu'il permet de recevoir, un Tuner peut permettre la réception de la radio FM (87.5 MHz – 108 MHz, pas de 0.1 MHz), de la radio AM (535 Hz – 1700 Hz, pas de 5 Hz), de la TV hertzienne (54 MHz – 88 MHz, pas de 0.01 MHz), etc...
On veut créer une classe `Radio` et une classe `TvRadio`, essayez d'utiliser au mieux les interfaces et les classes abstraites pour spécifier les différents Tuner que pourraient utiliser de tels terminaux de réception.