Université Paris Cité

Compléments en Programmation Orientée Objet
TP n° 5 : Mécanismes et stratégies d'héritage (TP noté)

Tous les documents (cours, web) sont autorisés. Le rendu se fait via Moodle.

Modélisation les polynômes : limites de l'héritage et du sous-typage

Important : Testez vos programmes (dans une méthode `TestExo.main()`) après chaque modification ! Pour ce faire, munissez toutes vos classes d'un `toString()` suffisament informatif.

En mathématiques, un polynôme, sur l'anneau unitaire [1] A, d'indéterminée X, est un objet de la forme $a_0 + a_1X + a_2X^2 + \cdots + a_nX^n$, où les coefficients a_i sont pris dans A. On note $A[X]$ l'ensemble de ces objets.

En programmation, on peut représenter un élément de $A[X]$ par la liste de ses coefficients a_i... et son plus grand degré n. Dans ce TP, nous supposons que $A = \mathbb{N}$.

Exercice 1 : Polynôme et Polynôme unitaire

Un *polynôme unitaire*, ou polynôme monique, est un polynôme dont le coefficient du terme de plus haut degré est égal à 1. C'est à dire que $a_n = 1$.

1. Programmez une classe `Poly`. Les attributs sont privés, accessibles par "getteurs" et les "setteurs".
2. Nous avons vu qu'un polynôme unitaire est un polynôme avec $a_n = 1$. En POO, la relation "est un" se traduit habituellement par de l'héritage.
   Programmez donc la classe `Upoly` qui étend `Poly` tout en garantissant que l'objet obtenu représente bien un polynôme unitaire.
   Vous avez dû trouver une solution sans ajouter d'attributs. Si ce n'est pas le cas, corrigez l'exercice précédent.
3. Créez un objet `c` de `Upoly` et modifiez a_n en utilisant le setteur de `Poly`.
   L'objet `c` est-il pourtant encore de type `Upoly` ?
4. Pour corriger ce problème, redéfinissez (avec `@Override`) les setteurs dans la classe Upoly de sorte à préserver l'invariant "cet objet représente un polynôme unitaire".
   Est-ce que cela résout vraiment le problème ?
5. Expliciter les propriétés des différentes classes (notamment le lien entre pré et post-conditions pour les mutateurs des coefficients)

Exercice 2 :

Comme le problème de l'exercice 1 est qu'on ne peut pas concilier à la fois l'invariant "cette polynôme est un XXX" et la spécification des mutateurs héritée du supertype, une version immuable des polynôme devrait être plus robuste.
Évidemment, les mutateurs fournissaient une fonctionnalité utile. Pour les remplacer, il est possible d'écrire des méthodes retournant un nouvel objet identique à `this`, sauf pour la propriété qu'on souhaite "modifier".

1. Ensemble muni d'une addition, d'une multiplication, toutes deux munies d'éléments neutres... avec quelques autre propriétés. Lire la page Wikipédia pour une définition complète.

À faire :

1. Écrivez les classes immuables `PolyImmuable` et `UpolyImmuable` sous-classes de `Poly` (classe abstraite fournissant les fonctionnalités communes, sans mutateur). Notez que les méthodes de "modification" de `UpolyImmuable` respectent automatiquement l'invariant du polynôme unitaire car `this` n'est pas modifié.
2. Ci-dessus, pour empêcher la création de sous-types mutables, il faudrait que `PolyImmuable` ne soit pas extensible ; mais dans ce cas, `UpolyImmuable` ne peut pas en être un sous-type.

   La technique des classes scellées permet, pour une classe donnée, de définir une liste fermée de sous-types et donc de garantir l'immuabilité du supertype. Utilisez cette technique pour définir des types immuables `Poly` et `Upoly` avec `Upoly` sous-type de `Poly` (imbriquez vos déclarations dans une classe-bibliothèque `RingImmuables`).