

## UE Conception Orientée Objet

### Expressions

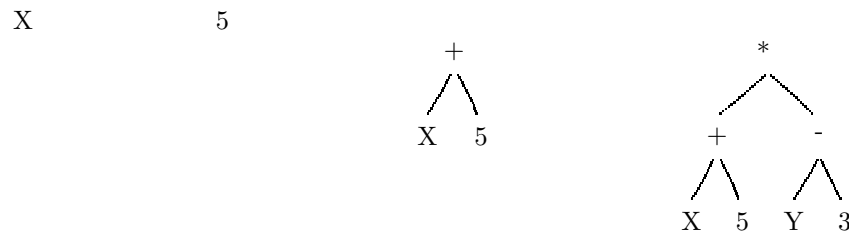
On s'intéresse à la modélisation d'expressions numériques symboliques définies ainsi :

Une *expression* est soit une *expression atomique* soit une *expression binaire*.

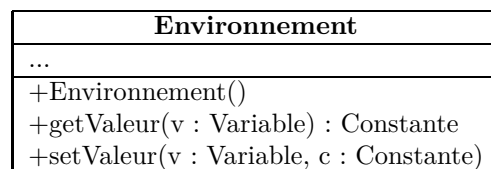
Dans les expressions atomiques on distingue les *constantes* et les *variables*. Ces expressions sont respectivement gérées par des classes nommées **Constante** et **Variable**.

Dans les expressions binaires on trouve les *somme*, *différence*, *produit* et *division* de deux expressions.

Voici des exemples de telles expressions représentées classiquement sous forme d'arbre (de gauche à droite on trouve une expression "variable", une expression "constante" et deux expressions binaires) :



Dans ce problème nous nous limiterons aux constantes à valeur entière (**int**). Les variables sont caractérisées par un identificateur qui sera une chaîne de caractères. Les variables peuvent se voir attribuer une valeur (constante). Ces valeurs sont mémorisées dans la classe **Environnement** dont voici le diagramme UML :



La méthode `getValeur` lève une exception `VariableNonLieeException` si aucune valeur n'est définie pour la variable passée en paramètre.

**Q 1 .** Donnez le code JAVA pour la classe `Environnement`.

Les opérations (méthodes) qu'il est possible d'invoquer sur un objet de type *expression* sont les suivantes : `public void afficher()` affiche l'expression en utilisant une notation infixée totalement parenthésée.

Ainsi pour les expressions des arbres ci-dessus on a respectivement l'affichage : `X`, `5`, `(X+5)`, `((X+5)*(Y-3))`.

`public Constante evaluer(Environnement e) throws VariableNonLieeException` évalue l'expression en fonction des valeurs des variables définies dans l'environnement *e*. Le résultat est une expression constante.

Le mécanisme d'évaluation est naturel : les constantes s'évaluent en elle-même, les variables s'évaluent en la valeur qui leur est associée dans l'environnement et les expressions binaires s'évaluent en appliquant leur opérateur (+, \*, - ou /) au résultat de l'évaluation de leurs 2 sous-expressions.

Une exception `VariableNonLieeException` si une variable n'a pas de valeur associée dans l'environnement fourni.

**Q 2 .** Proposez, sous forme de diagramme UML, un schéma de conception pour les types (classes et/ou interfaces) permettant la représentation de telles expressions.

Vous détaillerez les attributs et les méthodes et ferez clairement apparaître les éventuelles méthodes abstraites et surcharges de méthodes.

**Q 3 .** Donnez le code JAVA de la classe permettant la représentation d'expressions binaires de type somme. Vous donnerez également le code des "super-types" (interfaces et/ou superclasses<sup>1</sup>) nécessaires dont dépend cette classe.

**Q 4 .** La solution mise en place actuellement ne permet qu'un type d'affichage des expressions : l'affichage infixé totalement parenthésé. Il est cependant possible d'envisager d'autres affichages pour les expressions : postfixé, préfixé, etc.

Faites une proposition de conception élégante permettant de modifier facilement le type d'affichage pour les expressions (il sera nécessaire d'envisager de légères modifications sur la solution de conception précédente sur les expressions.).

**Q 5 .** Donnez le code JAVA correspondant à cette solution pour les 3 types d'affichage mentionnés.

<sup>1</sup>sauf `Object` évidemment !