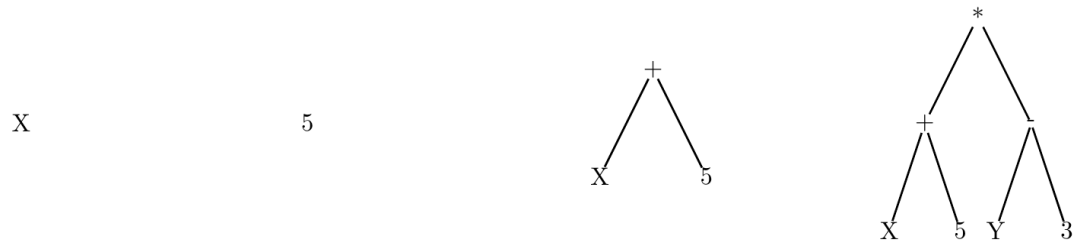
UE Conception Orientée Objet

Expressions

On s'intéresse à la modélisation d'expressions numériques symboliques. Une *expression* est soit *atomique* soit une *binaire*. Dans les expressions atomiques on distingue les *littéraux* (les valeurs constantes, *literal value*) et les *variables* (des identificateurs auxquels on peut lier une valeur littérale). Dans les expressions binaires on trouve les habituelles opérations *somme*, *différence*, *produit* et *division*.
Voici des exemples de telles expressions représentées classiquement sous forme d'arbre. De gauche à droite on trouve une expression variable, une expression littérale et deux expressions binaires :



Dans ce problème nous nous limiterons aux constantes à valeur entière. Les variables sont caractérisées par un identificateur (le « nom » de la variable). Les variables peuvent se voir attribuer une valeur (entière). Les liens entre les variables et leurs valeurs sont mémorisées dans une table appelée « environnement » (*environment*). La classe `Environment` permet de représenter ces environnements.

Environnement
- bindings : Map<Variable, Value>
+Environnement() +getValue(v : Variable) : Value +setValeur(v : Variable, c : Value)

où :

- `setValue(Variable, Value)` permet d'associer une valeur à une variable ;
- `getValue(Variable)` permet d'obtenir la valeur liée à une variable.

  Cette méthode lève l'exception `UnboundVariableException` si la variable en paramètre n'a pas de valeur associée dans cet environnement.

Parmi les méthodes qu'il est possible d'invoquer sur un objet de type *expression*, on trouve en particulier :

`format()` renvoie une chaîne de caractères représentant l'expression en utilisant une notation infixée totalement parenthésée.

Pour les expressions des arbres ci-dessus on obtient donc respectivement les chaînes : `"X"`, `"5"`, `"(X+5)"` et `"((X+5)*(Y-3))"`.

`eval(Environnement)` évalue l'expression en fonction des valeurs des variables définies dans l'environnement fourni en paramètre. Le résultat est une valeur littérale.

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

Q 1 . Proposez, sous forme de diagramme UML, un schéma de conception pour les types permettant la représentation de telles expressions.

Q 2 . Donnez le code Java de la classe permettant la représentation d'expressions binaires de type somme.

Q 3 . La solution mise en place actuellement ne permet de générer qu'un type de formatage des expressions : celui correspondant à une notation infixe totalement parenthésée. Il est cependant possible d'envisager d'autres formatages pour les expressions : postfixe, préfixe, etc.

Modifiez la solution précédente pour établir une proposition de conception qui permette d'utiliser plusieurs types d'affichage pour les expressions.

Il doit être possible d'éxécuter un code ressemblant à :

```
ExpressionFormater formater = ... ;
Expression exp = ...;
String result = formater.formatExpression(exp);
```

Q 4 . Donnez le code JAVA correspondant à cette solution pour au moins deux des types de formatages mentionnés.

Q 5 . Comment la solution de conception que vous avez appliquée pour le formatage peut-elle être reprise pour calculer l'évaluation des expressions ?

Utilisez alors cette solution pour mettre en œuvre un évaluateur d'expression qui utilise la valeur `0` dans le cas d'une variable non liée au lieu de lever une exception.