UBO – UFR Sciences – département d'informatique
Master TIIL-A – 2ème année

Métaprogrammation

kerboeuf@univ-brest.fr

1. Modèles et métamodèles en Java

Exercice : expressions arithmétiques

Une expression arithmétique est soit un nombre (*e.g.* « 3,5 »), soit une variable (*e.g.* « a »), soit un terme (*e.g.* « 3,5 + a») qui représente un opérateur appliqué à 2 autres expressions arithmétiques appelées opérandes. Un terme peut être une addition, une soustraction, une multiplication ou une division.

A faire : créer les classes Expression, Nombre, Variable, Terme, Addition, Soustraction, Multiplication et Division conformément à l'énoncé. Au final, les classes doivent pouvoir être utilisées ainsi :

```
Expression e1 = new Division(new Nombre(7), new Nombre(2)); //  (7 / 2)
Expression e2 = new Addition(e1, new Variable('a'));        // ((7 / 2) + a)
```

A faire : redéfinir les méthodes toString de manière à produire l'affichage suivant :

```
System.out.println(e2); // affichage de "((7.0 / 2.0) + a)"
```

Une expression peut être *évaluée*. L'évaluation d'une expression produit une autre expression éventuellement *réduite*. Par exemple, l'évaluation de « 3,5 », « a » et « 3,5 + a » donne les expressions identiques « 3,5 », « a » et « 3,5 + a». En revanche, l'évaluation de « 7 / 2 » et « (7 / 2) + a » donne les expressions réduites « 3,5 » et « 3,5 + a ».

A faire : définir les méthodes eval de manière à produire les résultats suivants :

```
Expression re1 = e1.eval(); // 3,5
Expression re2 = e2.eval(); // 3,5 + a
```

Un registre d'évaluation est un tableau associatif (*map*) qui associe une variable (type Character) à une expression (type Expression). Ce registre peut être fourni à la fonction eval afin que les variables soient remplacées par les valeurs correspondantes lors de l'évaluation.

A faire : définir les nouvelles méthodes eval de manière à produire les résultats suivants :

```
Map<Character, Expression> registre = new HashMap<>(); // registre d'éval.

registre.put('a', new Nombre(1.5));    // a -> 1,5
Expression re2bis = e2.eval(registre); // 5

registre.put('a', new Variable('b'));  // a -> b
Expression re2ter = e2.eval(registre); // 3,5 + b
```

A faire : définir le modèle UML du code final des expressions arithmétiques.

A ce stade, une expression arithmétique doit être définie à la main dans le code source pour pouvoir être évaluée.

A faire (si possible) : mettre en œuvre une solution à base d'XML pour permettre au programme de charger des expressions arithmétiques, de les évaluer, puis de les sauvegarder.

Exercice : machine à pile

Une machine à pile est une machine à calculer munie d'une pile de valeurs flottantes, d'un registre qui associe des variables à des valeurs, et d'une liste d'instructions. Les instructions peuvent être :

- push *p* : met *p* en sommet de pile (*p* peut être un nombre ou une variable)
- add : dépile 2 valeurs et place l'addition de ces valeurs en sommet de pile
- sous : dépile 2 valeurs et place la soustraction de ces valeurs en sommet de pile
- mult : dépile 2 valeurs et place la multiplication de ces valeurs en sommet de pile
- div : dépile 2 valeurs et place la division de ces valeurs en sommet de pile

A faire : créer les classes Machine, Instruction, Push, Add, Sous, Mult et Div. La classe Machine doit disposer d'une méthode toString qui indique l'état des trois composantes de la machine (pile, registre et instructions), d'une méthode store(*c*,*v*) qui associe *v* à *c* dans le registre, et d'une méthode input(*i*) qui rajoute l'instruction *i* à la liste d'instructions. Au final, les classes doivent pouvoir être utilisées ainsi :

```
Machine m = new Machine();
m.store('a', 1.5);
m.input(new Push(7)); m.input(new Push(2)); m.input(new Div());
m.input(new Push('a')); m.input(new Add());
System.out.println(m);
// PILE:[] REGISTRE:{a=1.5} INSTRUCTIONS:[PUSH 7, PUSH 2, DIV, PUSH a, ADD]
```

Une machine à pile peut être exécutée. Ses instructions sont alors exécutées dans l'ordre et supprimées au fur et à mesure. La machine s'arrête quand elle n'a plus d'instruction à exécuter.

A faire : définir les méthodes *run* de chaque instruction comme énoncé. Définir ensuite la méthode *run* de la machine à pile qui enchaîne les instructions jusqu'à ce qu'elles aient toutes été traitées. Exemple :

```
m.run();
System.out.println(m);
// PILE:[5.0] REGISTRE:{a=1.5} INSTRUCTIONS:[]
```

A faire : définir le modèle UML de la machine à piles.

Exercice : expression vers machine à pile

A faire : définir la fonction suivante qui prend en paramètre une machine à pile et une expression arithmétique, et qui transforme l'expression en série d'instructions enregistrées dans la machine :

```
public static void exp2map(Machine m, Expression e) { … }
```

Exemple d'utilisation :

```
Expression e1 = new Division(new Nombre(7), new Nombre(2)); //  (7 / 2)
Expression e2 = new Addition(e1, new Variable('a'));        // ((7 / 2) + a)

Machine m = new Machine();
m.store ('a', 1.5);
exp2map(m, e2);

System.out.println(m);
// PILE:[] REGISTRE:{a=1.5} INSTRUCTIONS:[PUSH 7, PUSH 2, DIV, PUSH a, ADD]
```

La transformation d'une expression en série d'instructions de machine à pile est faite par la *fonction* exp2map qui réalise des *traitements par cas* sur les types de données.

A faire (si possible) : mettre à jour le code des expressions arithmétiques afin que la transformation soit faite sans tests de types par la *méthode* toInst(m) déclarée dans Expression.