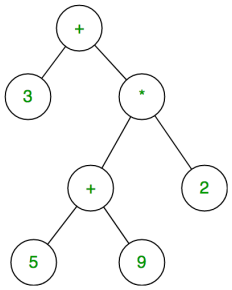

M1 GIL – Mini-projet d'architecture logicielle 2022–2023
Une bibliothèque de manipulation d'expressions symboliques.

F. Nicart

À rendre avant le **8 mai 2023**

1 Présentation générale du mini-projet

On souhaite disposer d'une bibliothèque permettant de manipuler des expressions quelconques de façon symbolique. C'est-à-dire que la construction se fera de façon purement syntaxique, sans donner de sémantique aux objets. Cependant, les expressions pourront être de types différents. On considérera les types suivants :

— les expressions arithmétiques,
— les fonctions,
— les expressions rationnelles.

De nouveaux types pourront être ajoutés à la bibliothèque dans le futur. Pour simplifier, on considérera que tous les types d'expressions pris en charge par la bibliothèque auront des opérateurs d'arité au plus 2. On produira des programmes de manipulation :

— Un éditeur d'expressions capable de procéder à la saisie d'expressions et de les charger ou de les sauvegarder dans un fichier au format XML.
— un programme permettant d'évaluer des expressions arithmétiques ou des fonctions.
— un programme permettant de dire si une expression rationnelle reconnaît le mot vide.

2 Spécifications

2.1 Les expressions

Les expressions seront composées d'opérateurs, unaires ou binaires, de constantes et de variables. Tous ces éléments sont purement symboliques et dépendent du type d'expression. Des expressions de types différents ne peuvent être combinées. Chaque type d'expressions possèdera un identifiant symbolique (une chaîne de caractères sans espace) permettant de faire référence à ce type dans les programmes utilisant la bibliothèque.

2.1.1 Le type « Expression arithmétique »

Ce type permet de construire des expressions représentant un calcul arithmétique, sans variable :

— Les opérateurs unaires consisteront en seulement l'opérateur `négatif`, représenté par le symbole "∼" ;
— Les opérateurs binaires consisteront en l'addition, la soustraction, la multiplication et la division représentés respectivement par les symboles "`+`", "`-`", "`*`" et "`/`" ;
— les constantes consistant en la représentation symbolique de tout nombre réel (correspondant au type `double`). Par exemple "`3.14e-10`" ;
— ces expressions ne comportent pas de variables.

Ce type sera identifié dans les programmes par le symbole "`arith`".

2.1.2 Le type « fonction »

Ce type permet de construire des expressions représentant un calcul arithmétique autorisant une seule variable :

— Les opérateurs unaires sont identiques à ceux des expressions arithmétiques.
— Les opérateurs binaires sont identiques à ceux des expressions arithmétiques.
— les constantes sont identiques à celles des expressions arithmétiques.
— les variables consistent en l'unique variable représentée par le symbole "`x`".

Ce type sera identifié dans les programmes par le symbole "`function`".

2.1.3 Le type « Expression rationnelle »

Ce type permet de construire des expressions dénotant des langages (ensembles de mots) à l'aide d'opérateurs rationnels :

— Les opérateurs unaires consisteront en seulement l'étoile de Kleene représentée par le caractère "$*$" ;
— Les opérateurs binaires consisteront en l'union et la concaténation représentées respectivement par les symboles "`+`", "$\cdot$" ;
— les constantes seront des mots composés sur l'alphabet $[a-z]$, de longueur comprise entre 1 et 20 ou du `mot vide` représenté par le symbole "`1`" ;
— ces expressions ne comportent pas de variables.

Ce type sera identifié dans les programmes par le symbole "`rational`".

2.2 Les programmes

2.3 L'éditeur `expedid`

Ce programme permet de choisir un type d'expressions, de saisir une expression sur ce type et sauvegarder ou charger le résultat depuis un fichier. L'éditeur fonctionnera en ligne de commande à la manière d'un interpréteur très simple. La saisie des expressions se fera selon la notation polonaise inverse (de façon post-fixe). L'éditeur disposera pour cela d'une pile contenant les expressions déjà saisies. Cette pile sera manipulée en fonction des saisies de l'utilisateur. Si ce dernier saisi un symbole correspondant à :

— une constante : une expression correspondant à cette constante est construite et placée au sommet de la pile ;
— une variable : une expression correspondant à cette variable est construite et placée au sommet de la pile ;
— un opérateur unaire : l'expression au sommet de la pile est dépilée puis combinée à l'opérateur correspondant à la saisie. L'expression résultante est placée au sommet de la pile ;
— un opérateur binaire : les deux expressions au sommet de la pile sont dépilées puis combinées à l'opérateur correspondant à la saisie. L'expression qui était au sommet de la pile correspondra à l'opérande droit, la seconde à l'opérande gauche. L'expression résultante est placée au sommet de la pile ;

Une erreur sera affichée si le nombre d'éléments dans la pile est insuffisant pour réaliser l'opération et la pile restera inchangée. De même si un opérateur est utilisé pour combiner des expressions de types différents.

L'interpréteur acceptera également des commandes en entrée. Afin de distinguer ces dernières des éléments potentiels d'un type d'expressions, le nom des commandes sera préfixé par un point d'exclamation :

— `!quit` : termine le programme.
— `!type [symbole du type]` : sans argument, affiche la liste des symboles des types d'expressions supportés par l'application. Avec un argument, utilise ce dernier pour sélectionner le type d'expression qui sera utilisé pour les commandes suivantes.
— `!save nom_fichier` : enregistre dans le fichier dont le chemin est indiqué la sérialisation en XML de l'expression au sommet de la pile. L'expression reste sur la pile.
— `!load nom_fichier` : désérialise le contenu XML du fichier dont le chemin est indiqué pour construire une expression. L'expression construite est placée au sommet de la pile.

Le contenu de la pile sera automatiquement affiché après chaque commande suivi du type d'expression en cours. La pile sera affichée de la manière suivante :

— du bas vers le sommet de la pile,
— chaque élément de la pile sera numéroté de $n-1$ à 0, 0 correspondant au sommet,
— le type de chaque élément sera affiché entre crochet,
— puis l'expression complète sera affichée sur une seule ligne en notation polonaise inverse.

Par exemple :

```
Stack is empty.
 [arith]
```

```
> 12
0 : [arith] 12.0
 [arith]
> 33
1 : [arith] 12.0
0 : [arith] 33.0
 [arith]
> 6.66
2 : [arith] 12.0
1 : [arith] 33.0
0 : [arith] 6.66
 [arith]
> *
1 : [arith] 12.0
0 : [arith] 33.0 6.66 *
 [arith]
> +
0 : [arith] 12.0 33.0 6.66 * +
 [arith]
> !quit
```

2.4 Le programme `calc`

Ce programme fonctionne également en ligne de commande avec la syntaxe suivante :

```
java calc filename [value]
```

Le premier paramètre correspond au chemin d'un fichier contenant une expression soit de type arithmétique (`arith`), soit de type fonction (`function`). Le second paramètre doit être fourni uniquement dans le second cas uniquement et correspondra à la valeur affectée à la variable "`x`" pour le calcul. Le programme affichera le résultat de l'évaluation de l'expression dans le terminal le résultat avant de se terminer. Tout autre type d'expression déclenchera une erreur.

Note : on pourra fournir un script shell permettant de l'invoquer avec la syntaxe :

```
calc filename [value]
```

2.5 Le programme `nullable`

Ce programme fonctionne également en ligne de commande avec la syntaxe suivante :

```
java nullable filename
```

L'unique paramètre correspond au chemin d'un fichier contenant une expression de type rationnelle (`rational`). Le programme affichera dans le terminal le texte "`vrai`" si le langage de l'expression contient le mot vide, "`faux`" sinon. Tout autre type d'expression déclenchera une erreur.

3 Contraintes de réalisation

- Il ne doit pas être possible de pouvoir combiner des expressions de types différents.
- La bibliothèque doit pouvoir être utilisée dans des programmes manipulant des expressions en dur.
- Il n'est pas demandé qu'il soit possible d'ajouter des nouveaux types d'expressions sans recompiler les programmes.

4 Modalités

- Le travail est à réaliser en binôme maximum [1].
- L'implémentation et le compte-rendu sont à rendre avant le **8 mai 2023** [2].

1. Les singletons sont acceptés ;-). Ceci dit le travail en binôme est recommandé pour la confrontation d'idées lors de la conception. Un niveau d'exigence plus élevé sera appliqué aux trinômes, en particulier lorsqu'ils ne se sont pas déclarés au début du projet.
2. Des pénalités de retard seront naturellement appliquées.

5 Travail à rendre

Le projet est à déposer sur Universitice. Vous fournirez une archive nommée `projet-al-nom1-nom2.zip` comportant un dossier éponyme. Celui-ci contiendra le code source de votre implémentation ainsi que votre rapport au format `PDF`.

Il est important de noter que le rapport aura autant d'importance que l'implémentation. Celui-ci devra contenir :

- Une explication de l'architecture globale illustrée par un ou plusieurs diagrammes UML.
- Pour chaque patron mis en œuvre : la motivation du choix du patron, le diagramme UML de la partie de votre application correspondant au patron instancié, en prenant soin d'indiquer pour chaque acteur du patron l'entité (interface/classe) qui lui correspond.
- Éventuellement, une liste des patrons que vous avez hésité à utiliser et les raisons pour lesquelles vous les avez écartés.
- Une DTD pour votre format XML.
- Un bilan du respect des grands principes d'architecture vus en cours.
- Le moins de fautes d'orthographe et de français possible. (Relisez-le ! Faites-le relire…)

Une attention particulière sera portée à la précision sémantique des noms de vos entités (à vos dictionnaires d'anglais !), ainsi qu'à l'architecture des paquetages.

Un autre point **important** à garder à l'esprit est que ce mini projet est un prétexte à l'élaboration d'une architecture de taille raisonnable mais respectant les bons principes vus en cours. Par conséquent, le logiciel n'a pas vocation à être abouti ni ergonomique. On pourra ainsi utiliser des bouchons pour les fonctions pour lesquelles le temps a manqué en le mentionnant dans le rapport et en décrivant le travail à réaliser pour compléter la fonctionnalité, en particulier pour une fonctionnalité pour laquelle une autre fonctionnalité a été développée qui utilise le même mécanisme/patron. Par exemple, les deux programmes d'évaluation peuvent être considérés comme redondants entre-eux.

6 Quelques indications

- Vous soignerez la modularité de votre application et justifierez dans votre rapport la répartition en paquetages.
- (Comme d'habitude,) vous soignerez la rédaction de vos interfaces. Vous détaillerez leur conception dans votre rapport.
- Vous fournirez, en annexe du rapport, la liste complète, par paquetage, des entités de votre application. Vous donnerez en face du nom de chaque entité sa responsabilité dans l'application en une phrase.
- Vous produirez la javadoc de l'ensemble de votre application que vous fournirez, après l'avoir lue, dans un sous-dossier intitulé `javadoc` de votre archive.