Gabriel Couroux

Rapport du TP n°4 de Programmation Orientée Objet

Les choix

Ce problème a été découpé en trois grandes parties.

Première partie : Consiste en la vérification de l'intégrité de l'équation passé en paramètre. Dans tous les cas indésirable tel qu'un String vide, qui contient des caractères qui ne fond pas partie d'une équation, des problème avec des parenthèses ou avec des signes d'opération ; on renvoie une exception de type ArithmeticException.

Seconde partie : Après s'être assuré que l'équation soit bien formée, on procède à sa transformation en un objet lisible par l'interpréteur. La plus grande difficulté fut de trouver les règles de construction pour préserver les priorités de calcule entre les opérations et les parenthèses lors de la lecture par l'interpréteur. Ces règles devaient être suffisamment générales pour que n'importe qu'elle type d'équation puis être résolue, même les plus longues et les plus complexes.

Les principaux algorithmes

En dehors du package principal et du package Interpréteur, le double cœur du programme réside dans les packages Analyse et Transformation. Chacun ne comporte qu'une seule classe. La première classe Analyse vérifie l'intégrité de l'équation grâce à quatre méthodes :

➔**verifEquation** renvoie une exception lors que l'équation est vide, ou quelle contient des caractères indésirables.

➔**VerifParenthese** renvoie une exception lors que des parenthèses sont orphelines, ou lors que une opération se retrouve à gauche d'une parenthèse fermée ou qu'un '*' ou un '/' se trouve à droite d'une parenthèse ouverte.

➔**VerifOperations** renvoie une exception lors que deux opération se suivent, s'il existe des '*' implicites ou qu'une opération ne se trouve pas au bonne endroit

➔**suppEspaces**, raccourcie l'équation en suppriment tous les espaces présent dedans.

La seconde classe Constructeur produit un arbre binaire grâce au design pattern composite, les expressions non terminales sont toujours des opérations et les expression terminales toujours des nombres. À l'aide de **Construction** on forme l'expression Lisible de façon récursive. On commence par enlever toutes les parenthèses inutiles puis on vérifie si l'équation est Terminal, avec **queDesChiffres**. Sinon on cherche l'**operationLaMoinsImpotante** de l'équation en évitant toutes les parenthèses avec **passerParentheses**. Pour finir on construit une NonTerminal avec l'opération trouvée comme valeur et construit deux nouvelles expression pour le fils gauche et le fils droit, avec les éléments de chaque côtés de l'opération.

Patrons de conception

Deux designs patterns sont utilisés, le design pattern composite et le design pattern interpréteur. Leur structure étant similaire, dans notre cas les deux designs patterns sont presque entièrement confondu. Le composite s'articule autour de la classe abstraite Expression et de ses deux filles Terminal et NonTerminal. L'interface de l'interpréteur est confondue avec la classe abstraite Expression. Le composite permet la construction d'un objet plus facilement lisible par l'interpréteur grâce à sont arborescence.

Diagramme UML