Interface et classe abstraite

Exercice 1 - Arbre d'expressions

Le but de cet exercice est de construire un parseur d'expressions arithmétiques simples. Ces expressions sont représentées sous forme d'arbres.
L'ensemble des classes devront être définies dans le paquetage `fr.umlv.calc` si aucun paquetage n'est indiqué.

La classe `OpOrValue` définit des objets qui peuvent être soit des valeurs soit des opérations (avec un membre gauche et un membre droit).

1. Pourquoi le constructeur qui prend 4 arguments est déclaré `private` ?
2. Quel est le problème du constructeur `OpOrValue(int, OpOrValue, OpOrValue)` ?
   Corriger le problème.
3. Écrire une méthode `parse` qui prend un `Scanner` en entrée et crée l'arbre d'expression correspondant sachant que l'arbre sera donné au scanner en utilisant la notation polonaise inverse. Par exemple, au lieu de `2 + 3 - 4`, la notation polonaise inverse est `- + 2 3 4`.

   Indication: la méthode `parse` est naturellement récursive: si l'expression contient encore des symboles (et qu'elle est bien formée) alors:
   - soit le prochain symbole est un entier et il suffit d'en faire une feuille de l'arbre d'expression,
   - soit le prochain symbole est un opérateur et il faut appeler `parse()` 2 fois pour obtenir le fils gauche et le fils droit et les combiner avec l'opérateur pour faire une nouvelle expression.

   Enfin, pour rappel, `scanner.next()` renvoie le prochain mot, `Integer.parseInt()` permet de savoir si c'est un entier et il est possible d'utiliser le `switch expression` (celui avec des flèches) sur des Strings en Java.
   Note: pour activer le `switch expression`, vous devez cocher "enable preview feature for Java 13" dans Java > Compiler dans les propriétés du projet.
4. Déplacer le `main` dans une nouvelle classe `Main` dans le package `fr.umlv.calc.main`.
5. Écrire la méthode d'affichage de l'arbre d'expression pour que l'affichage se fasse dans l'ordre de lecture habituel.
   Note: il va falloir ajouter des parenthèses !
6. Noter que prendre un `Scanner` en paramètre ne permet pas de ré-utiliser la méthode `parse` si, par exemple, l'expression à parser est stockée dans une `List` de `String`.
   Quelle interface que doit-on utiliser à la place de `Scanner` pour que l'on puisse appeler la méthode `parse` avec une `List` ou un `Scanner`.

Exercice 2 - Arbre d'héritage

La classe `OpOrValue` est en fait mal codée, car elle n'est pas codée en objet. En effet, chaque objet stocke trop d'information et la distinction entre le type valeur et le type opération se fait par un champs et non pas par la classe elle-même.

1. Écrire les classes `Expr`, `Value`, `Add` et `Sub` ayant les mêmes fonctionnalités que la classe `OpOrValue`, mais codée de façon objet.
   De plus, modifier la classe `Main` de test.

2. Dans quelle classe doit-on mettre la méthode `parse` ? Modifier celle-ci pour qu'elle marche avec les nouvelles classes.
3. Écrire une méthode `eval` calculant le résultat de l'expression.
4. Discuter sur le fait de transformer la classe `Expr` en classe abstraite ou en interface. Faire les changements qui s'imposent.
5. On peut noter que les classes `Add` et `Sub` ont beaucoup de code commun... Comment re-factoriser le code pour mettre le code commun à un seul endroit?
   Quelles doivent être les visibilités des différentes méthodes ?
   Quelle doit être la visibilité de la classe introduite ?
   Rappel: les champs doivent toujours être privés !
