TD III

TD 03 : Notation Polonaise

III.1 Notation polonaise

Nous allons ici manipuler des expressions arithmétiques exprimées à l'aide de la notation polonaise inverse (NPI). La NPI, également connue sous le nom de notation post-fixée, permet d'écrire de façon non ambiguë les formules arithmétiques sans utiliser les parenthèses. Le reste du sujet, hormis les questions, est tiré de la page wikipédia francophone dédiée à cette notation: https://fr.wikipedia.org/wiki/Notation_polonaise_inverse

III.1.a Méthode pour apprendre la NPI facilement

La NPI peut être vue comme intuitive, sa difficulté relvant essentiellement d'un manque d'habitude (la plupart des calculatrices non HP ne l'utilisent pas). Pour traduire une expression algébrique (telle que par exemple $((1+2)\times 4)+3)$), il suffit de la lire en se disant ce que l'on doit faire, c'est-à-dire comprendre l'expression algébrique, faire les opérations dans le bon ordre (commencer dans l'excemple ci-dessus par l'addition de 1 et 2 , puis multiplier par 4 et enfin, ajouter 3).

Le calcul $((1+2)\times 4)+3)$ peut se traduire intuitivement par:

- je mets 1 (1);
- j'ajoute 2, (2+);
- je multiplie par 4 (4×);
- j'ajoute 3 (3+).

ce qui donne simplement 1 2 + 4 × 3 +.

L'équation 1 2 + 4 × 3 + peut aussi s'écrire 3 4 1 2 + × + avec des algorithmes de calcul équivalents.

Question 1.1
Quelle est la valeur de l'expression en NPI 3 4 1 2 + × + ?

III.1.b Calcul à l'aide de piles

Vous disposez des structures suivantes:

```
1 enum Type {
2     num,
3     op;
4 }
5
6 public class Fragment {
```

```
    Type type;
    String value;
    Fragment(Type type, String value) {...}
}
```

```
public class Pile {
    Pile() {...}

    boolean estVide() {...}

    /**
     * Dépile le fragment au sommet de la pile.
     */
    Fragment dépiler() {...}

    /**
     * Dépile un élément de la pile
     * @return l'élément dépilé si la pile n'est pas vide
     * @throws Exception "Pile vide" si la pile est vide
     */
    void empiler(Fragment f) {...}
}
```

On considère à présent la classe **Analyseur** ci-dessous:

```
class Analyseur {

/**
 * Évalue une expression en notation polonaise inversée (NPI)
 * @param expr expression en NPI sous forme de chaîne de caractères
 * @return le résultat de l'évaluation
 * @throws Exception "Error parsing expression" si l'expression ne peut pas êtr
    e parsée
 * @throws Exception "Not enough operands" si l'expression est incorrecte (
    manque d'opérandes)
 * @throws Exception "Too many operands" si l'expression est incorrecte (trop d
    'opérandes)
 * @throws Exception "Invalid operand format" si l'un des opérandes n'est pas
    un entier valide
 * @throws Exception "Division by zero" si une division par zéro est tentée
 * @throws Exception "Unknown operator" si l'expression contient un opérateur
    non reconnu
 */
    int eval(String s) {... }

/**
 * Analyse lexicale, découpage en fragments selon les opérateurs + - * /
 * Le format des entiers n'est pas vérifié ici
 * @param expr expression sous forme de chaîne de caractères
 * @return un tableau de fragments
 * @throws Exception "Invalid operator" si l'expression contient un opérateur
    non reconnu
 */
    Fragment[] parse(String s) {...}

/**
* Renvoie la valeur entière d'une chaîne de caractères représentant un entier
    naturel au format décimal
 * @param s chaîne de caractères représentant un entier naturel au format déc
    imal
 * @return la valeur entière correspondante
 * @throws Exception "Invalid integer format" si la chaîne n'est pas un entier
    naturel au format décimal valide
 */
```

```
31      int entier(String s) {...}
32  }
```

Question 1.2
Écrivez la méthode `eval` en supposant que les méthodes `parse` et `entier` sont déjà implémentées.

Question 1.3
Écrivez la méthode `entier`. Vous pouvez utiliser la méthode `charAt` de la classe `String` pour accéder aux caractères d'une chaîne. Notez que vous pouvez savoir si un caractère `c` est un chiffre en testant si $c >=' 0' \&\& c <=' 9'$. Vous pouvez obtenir la valeur entière d'un chiffre `c` en évaluant l'expression $c -' 0'$.

Question 1.4
Implémentez la classe `Pile`.

Question 1.5
Implémentez la méthode `parse`.

Il existe plusieurs stratégies pour résoudre ce problème:

- l'une d'elle est une approche récursive qui n'est pas demandée dans un premier temps (uniquement une fois que vous aurez terminé la deuxième approche)
- l'autre (celle qui est demandée prioritairement ici) consiste à ne se baser que sur des piles et peut être illustrée avec le schéma suivant:

équation	$((1+2)\times 4)+3)$						
Pile				2			
			1	1	3		
		4	4	4	4	12	
	3	3	3	3	3	3	15
entrée (String)	3	4	1	2	+	×	+