TP 5: Calculatrice

1 Calculatrice – Entiers

Donner une calculatrice qui permet d'evaluer des expression arithmétiques simples, utilisant **des entiers** et les opérations $+, -, *, /$. (**NB:** Ici, l'opération $/$ est la division entière, alors le résultat est aussi un entier.)

1. Les expressions seront données au clavier en notation polonaise inverse – RPN (Reverse Polish Notation). Ça veut dire que l'opération suit les opérands! Ex: L'expression $3\ 5\ +$ est égale à 8. (En notation classique $3+5=8$). L'expression $3\ 5\ +\ 4\ 1\ -\ *$ est égale à $(3+5)*(4-1)=24$.
2. Votre programme doit avoir une boucle principale qui
   - Maintient une pile de nombres (vous pouvez utilisez la classe `LinkedList` et l'interface `Deque`)
   - À chaque étape demande à l'utilisateur de donner un nombre ou une opération
   - Si l'utilisateur donne un nombre, le nombre est empilé.
   - Si l'utilisateur donne une opération, cette opération est appliquée aux deux nombres au sommet de la pile. Ces deux nombres sont depilés et remplacés par le résultat.
   - À chaque étape, la boucle affiche les contenus de la pile.
3. La boucle principale doit continuer jusqu'à le moment où l'utilisateur donne `Ctl-D` (End-of-File). (Astuce: vous pouvez le detecter avec la méthode `hasNext()` de la classe `Scanner`).

Si le programme rencontre un problème (par exemple, l'utilisateur donne un caractère inconnu, où il tente d'appliquer une opération alors que la pile ne contient pas de nombre), il faudra ratrapper les exceptions eventuelles, afficher un message avec des informations et puis terminer le programme.

Exemple d'utilisation :

```
Enter number, operation, or Ctl-D to exit
2
Current state:
[2]
3
Current state:
[2, 3]
4
Current state:
[2, 3, 4]
5
Current state:
[2, 3, 4, 5]
```

```
+
Current state:
[2, 3, 9]
7
Current state:
[2, 3, 9, 7]
-
Current state:
[2, 3, 2]
*
Current state:
[2, 6]
-
Current state:
[-4]
```

2 Calculatrice Polymorphique

Pour cette partie, vous devez modifier votre programme afin de traiter d'autres types de nombres. Le but de cet exercice est de donner une version du programme principale **polymorphique**, c'est-à-dire, une version qui peut fonctionner avec aucune modification (ou des modification très légéres) pour des types des données différentes. On demande de traiter

- Les nombres entiers (`Integer` ou `int`)
- Les nombres flottants (`Double` ou `double`)
- Les nombres rationnels (vous pouvez utiliser la class `Rational` du TP 2).

Afin de permettre à votre programme principale de traiter ces trois types de manière unifiée, il faudra construire une hiérarchie d'héritage. On propose de définir l'interface suivante :

```
1 interface Value {
2         Value add(Value t);
3         Value mult(Value t);
4         Value div(Value t);
5         Value sub(Value t);
6 }
```

Vous devez suivre les specification suivantes :

1. Donnez trois classes (`myInt`, `myDouble`, et `Rational`) qui **implementent** l'interface `Value`.
2. Écrivez une seule version de la boucle principale du programme et de la fonction d'évaluation des expressions en RPN, en utilisant l'interface. Notamment, votre pile doit maintenant contenir des éléments de type déclaré `Value`.

Exemple de l'utilisation du programme:

```
What type of calculator is this? 1=Int, 2=Double, 3=Rational
3
Enter number, operation, or Ctl-D to exit
1
Current state:
[(1/1)]
2
Current state:
[(1/1), (2/1)]
/
Current state:
[(1/2)]
2
Current state:
[(1/2), (2/1)]
3
Current state:
[(1/2), (2/1), (3/1)]
/
Current state:
[(1/2), (2/3)]
+
Current state:
[(7/6)]
```