

Écrire une classe `PileVector`, qui implémente l'interface `Pile` donnée ci-dessous, en utilisant comme structure interne de la classe un `java.util.Vector`<sup>1</sup> pour stocker les nombres réels (voir *annexe a*). L'interface `Pile` est la suivante :

Écrire un programme pour tester cette classe.

- les opérateurs binaires :  $+$ ,  $-$ ,  $*$ ,  $/$
- les opérateurs unaires : inverse et racine carrée notés `inv` et `rac`.

- **Pour chaque** composant de la chaîne (voir annexe c) :
  - s'il correspond à un **nombre**, on empile sa valeur ;
  - si c'est un **opérateur binaire**, on dépile deux valeurs puis on effectue l'opération et enfin on empile le résultat (attention à l'ordre des opérandes) ;
  - si c'est un **opérateur unaire**, on dépile une valeur puis on effectue l'opération et enfin on empile le résultat.
- **À la fin** du traitement, si la chaîne est bien formée il reste exactement une valeur dans la pile, c'est le résultat de l'évaluation.

- Une classe **Opérateur**. Cette classe comprendra :
  - une méthode de classe permettant de déterminer si l'opérateur (une chaîne) est bien un opérateur ;
  - une méthode booléenne permettant de déterminer si l'opérateur est unaire (il sera binaire dans le cas contraire) ;
  - deux méthodes de calcul, `calcul(Double a)` et `calcul(Double a, Double b)`, retournant un `Double` correspondant au résultat souhaité ou `null` si une erreur se présente.
- Une classe **Expression** où sera effectuée l'évaluation à l'aide d'une méthode `eval()`. On suppose que l'expression ne contient que des opérateurs ou des nombres correctement formés. Afficher un message d'erreur et quitter la méthode si l'expression est mal formée. Une expression est mal formée si :
  - un opérateur se situe à une mauvaise position : `rac 8`
  - l'expression est incomplète : `4.25 inv 8`
  - l'expression est vide.
- Une classe **Test**. La chaîne de caractères représentant l'expression est donnée en argument de la méthode `main`.

2011-2012

## ANNEXES

### a) Nombre réel et classe java

La classe `java.lang.Double` (ou `java.lang.Float`) permet de représenter le type primitif `double` (ou `float`) sous la forme d'une classe.

Ceci s'avère particulièrement utile en Java, où lorsque l'on manipule des objets (instances de classes) plutôt que des valeurs de types primitifs.

Bon à savoir :

```
float pi = 3.14;    // type primitif

// instances de la classe Double
Double f1 = new Double(3.14);
Double f2 = new Double(pi);

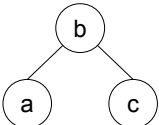
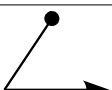

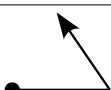
// conversion float --> Double
f1 = Double.valueOf(pi); // ou f1 = pi;

// conversion Double --> float
pi = f1.floatValue(); // ou pi = f1;

// conversion String --> Double
f1 = Double.parseDouble("3.14");

// conversion Double --> String
String s = f1.toString();
```

### b) Parcours préfixé, infixé et postfixé d'un arbre binaire

			
	parcours préfixé : b a c	parcours infixé : a b c	parcours postfixé : a c b

### c) Décomposer une chaîne de caractères

Pour décomposer une chaîne de caractères en éléments qui sont appelés « tokens » on peut utiliser la classe `java.util.StringTokenizer`. Par défaut, le caractère *espace* cloisonne les différents tokens. Ci-dessous un extrait de la documentation de cette classe :

```
StringTokenizer st = new StringTokenizer("this is a test");

String str;
while ( st.hasMoreTokens() )
{
    str = st.nextToken();
    System.out.println( str );
}

/*
    this prints the following output:
    this
    is
    a
    test
*/
```