IP1 2015–2016 DM (groupe Info 4)

Ce sujet est aussi disponible en ligne : http://www.pps.univ-paris-diderot.fr/~boutier/enseignement.html.

Vous devez soumettre le rendu sur Didel (cours IP1/travaux/DM Info 4) : http://didel.script.univ-paris-diderot.fr/claroline/work/work_list.php?assigId=54&cidReset=true&cidReq=IP1.

Seule la soumission la plus tardive sera corrigée.

Procédures et fonctions autorisées

Vous pouvez copier-coller (et utiliser) les procédures et fonctions suivantes dans votre code :

```
/** Renvoie le nombre de cases d'un tableau */
public static int getArrayLength(Object [] t) {
    return t.length;
}

/** Convertit un entier en chaîne de caractères (en base 10). */
public static String intToString(int n) {
    return "" + n;
}

/** Convertit une chaîne de caractères en un entier (relatif),
 * ou 0 si la chaîne ne représente pas un entier en base 10.
 */
public static int stringToInt(String s) {
    try {
        return Integer.parseInt(s);
    } catch(NumberFormatException e) {
        return 0;
    }
}

/** Affiche une chaîne de caractères sur le terminal. */
public static void showString(String s) {
    System.out.print(s);
}

/** Renvoie vrai si s1 et s2 sont égales. */
public static boolean areStringEqual(String s1, String s2) {
    if (s1 == null) {
        return s2 == s1;
    } else {
        return s1.equals(s2);
    }
}
```

Problème : un calculateur

Nous allons tenter d'implémenter un programme capable de résoudre des opérations, comme une calculette. Pour plus de simplicité au déboguage, nous afficherons les étapes faites par le calculateur. Les arguments sont donnés en ligne de commande, et arrivent donc au programme par un tableau de String. Voici un exemple d'utilisation :

```
$ java Calc 1 + 1 "*" 2 + 3 - 4 / 2
1 + 1 * 2 + 3 - 4 / 2
1 + 2 + 3 - 4 / 2
1 + 2 + 3 - 2
3 + 3 - 2
6 - 2
4
```

Le résultat est bien 4. Notez les guillemets autour de l'étoile ; ils sont nécessaires (l'étoile est une commande spéciale pour le terminal : il la remplace par plein de choses).

On supposera toujours que l'utilisateur utilise correctement le programme ! On ne traitera donc pas `$ java Calc + 1 "*" / 32`, par exemple. (Vous pouvez le faire, mais ça ne vous rapportera rien.)

Au commencement était la classe...

Créez un fichier Calc.java, dans lequel vous écrirez votre classe :

```
class Calc {

    /* Ajoutez ici vos fonctions et celles dont vous vous servirez */

    public static void main(String [] args) {
    }
}
```

Une opération, c'est déjà ça...

Objectif : un simple calcul, comme `1 + 1` ou `7 * 3`.

Reconnaître un opérateur.

Écrivez une fonction `public static boolean isOperator(String str)` qui renvoie `true` si et seulement si `str` est un opérateur, c'est à dire `+`, `-`, `*` ou `/`, et `false` sinon. Attention : utilisez `areStringEqual`, et pas `==`.

Une fonction de calcul.

Écrivez une fonction `public static String compute(String left, String operator, String right)` qui renvoie le résultat, sous forme de `String`, de "left" "operator" "right".

Indication : convertissez d'abord `left` et `right` en `int`, puis calculez le résultat en fonction de `operator`, puis convertissez le résultat en `String`.

La première version du programme !

On suppose que la ligne de commande donne exactement 3 arguments : deux nombres et un opérateur entre deux. Écrivez une procédure `public static void firstTry(String[] args)` qui calcule le résultat, puis l'affiche. Appelez cette fonction dans le main pour calculer le résultat de la ligne de commande.

Exemple : `$ java Calc 1 + 1` doit afficher `2`.

... plusieurs opérations, c'est bien...

Objectif : Plusieurs calculs, mais tous les opérateurs ont la même priorité, comme "1 + 1 + 4 - 3 + 5" ou "2 * 4 / 4 * 3".

Préparons-nous à déboguer.

Écrivez une procédure `public static void showArray(String[] t)` qui affiche les éléments de `t` séparés par une espace. Je ne veux pas d'espace à la fin, mais un retour à la ligne !

Exemple : `showArray({"1","+","3"})` doit afficher, sans espace après le 3 :

```
1 + 3
```

Compter les opérateurs.

Écrivez une fonction `public static int countOperators(String[] t)` qui renvoie le nombre d'opérateurs dans `t` (on utilisera `isOperator`).

Exemple : `countOperators(["1","+","3","−","5"])` renvoie 2.

Décaler un tableau.

Écrivez une procédure `public static String[] shiftArrayTwo(String[] t, int from)` qui renvoie un nouveau tableau égal à `t` décalé de deux cases vers la gauche, à partir de l'indice `from`.

Exemples : `shiftArrayTwo({1, 2, 3, 4, 5, 6}, 0)` renvoie `{3, 4, 5, 6}`.

`shiftArrayTwo({1, 2, 3, 4, 5, 6}, 1)` renvoie `{1, 4, 5, 6}`.

Deuxième version du programme !

Écrivez une procédure `public static void secondTry(String[] args)` qui, autant de fois qu'il y a d'opérateurs, fait le calcul du premier opérateur du tableau, stocke le résultat et décale le tableau de sorte que les 3 premiers éléments du tableau soient remplacés par le résultat de leur calcul. On affichera le contenu du tableau à chaque étape. Remplacez l'appel de `firstTry` par l'appel à `secondTry` dans le main.

Exemple :

```
$ java Calc 1 + 2 + 3 - 5
1 + 2 + 3 - 5
3 + 3 - 5
6 - 5
1
$ java Calc 1 + 2 "*" 3 - 5
1 + 2 * 3 - 5
3 * 3 - 5
9 - 5
4
```

Indication : faites simple, et n'utilisez qu'une seule variable pour les tableaux (args) ; en effet, on ne doit manier qu'un seul tableau à la fois.

...mais si on distingue les priorités, c'est mieux !

Priorité d'opérateur.

Écrivez une fonction `public static boolean hasHigherPriority(String op1, String op2)` qui renvoie vrai si op2 a une plus grande priorité que op1, et faux sinon. Remarquez que si op1 et op2 ont la même priorité, la fonction renvoie faux. (Le comportement de la fonction n'est pas défini si `op1` ou `op2` ne sont pas des opérateurs.)

Exemple : `hasHigherPriority("+", "*")` renvoie `true`, `hasHigherPriority("+", "-")` renvoie `false`.

Le premier à faire.

Écrivez une fonction `public static int searchMostPrioritizedOperator(String[] t)` qui renvoie l'indice du premier opérateur le plus prioritaire de `t`.

Exemple : `searchMostPrioritizedOperator(["1","+","3","*","5", "/" , "2"])` renvoie 3, ce qui correspond à `"*"`.

Indication : déclarez une variable correspondant à un indice, que l'on retournera à la fin de la fonction, puis parcourez le tableau à la recherche d'un opérateur (`isOperator`). Si cet opérateur est prioritaire (`hasHigherPriority`) à celui jusqu'ici retenu, mettez la variable à jour.

Troisième version du programme !

Écrivez une procédure `public static void thirdTry(String [] args)`, qui fait ce qu'on veut ! Remplacez l'appel de `secondTry` par l'appel à `thirdTry` dans le main.

Exemple :

```
$ java Calc 1 + 2 "*" 3 - 5
1 + 2 * 3 - 5
1 + 6 - 5
7 - 5
2
```

Mieux encore ?

Saurez-vous enrichir votre calculateur avec des parenthèses ?

Saurez-vous ajouter un opérateur "puissance" `^`, de plus forte priorité que tous les autres opérateurs ? (`2 * 2 ^ 3 -> 2 * 8 -> 16`)