

Initiation au développement - SDA : TP5

DUT/INFO/R1_01

version 2021-2022 (PN BUT 2021)

Table des matières

- 1. Exemple d'utilisation du Type Abstrait Pile : les EPF (Expressions Post-Fixées)
- 1.1. Exemple 1 :
- 1.2. Exemple 2 :
- 1.3. Exemple 3 :
- 1.4. Programme de résolution d'EPF :

2. Piles d'entiers

3. Exercice : ProgrammeCalcullette.java

4. Pour aller plus loin

Préambule

SI votre type abstrait **Pile** implémenté dans votre fichier **ProgrammePile.java** NE permet PAS de faire passer tous les tests de [PileTest.java](#)

ALORS utiliser ceux-ci : [ProgrammePile.java](#) et [Pile.java](#).

1. Exemple d'utilisation du Type Abstrait Pile : les EPF (Expressions Post-Fixées)

Bien, nous allons maintenant voir un exemple concret avec les piles.

- Une expression est dite post-fixée ou écrite en notation polonaise inversée lorsque chaque opérateur est immédiatement précédé de ses deux opérandes.
- Les expressions suivantes en sont des exemples :

```
4 2 +
3 2 * 5 4 * +
1 2 5 3 + * + 7 -
```

Commençons par évaluer manuellement ces 3 expressions.

Pour les évaluer, on applique la définition : chaque opérateur est précédé immédiatement de ses deux opérandes.

1.1. Exemple 1 :

- On parcourt l'EPF jusqu'au premier opérateur qui est + et on l'applique aux deux opérandes qui le précèdent, 4 et 2, ce qui donne au final 4+2=6. Bon, c'était le plus simple !

1.2. Exemple 2 :

- Lorsque l'expression devient plus compliquée, il est plus simple de faire un petit schéma pour comprendre. Pour faire ce schéma, on écrit tous les chiffres sur la même ligne. Ensuite, on parcourt à nouveau l'EPF jusqu'au premier opérateur et on l'applique aux deux opérandes juste avant. Et on continue jusqu'à ce que tous les opérateurs aient été traités. On peut les relier avec des flèches.
- Essayer de comprendre avec le dessin suivant :

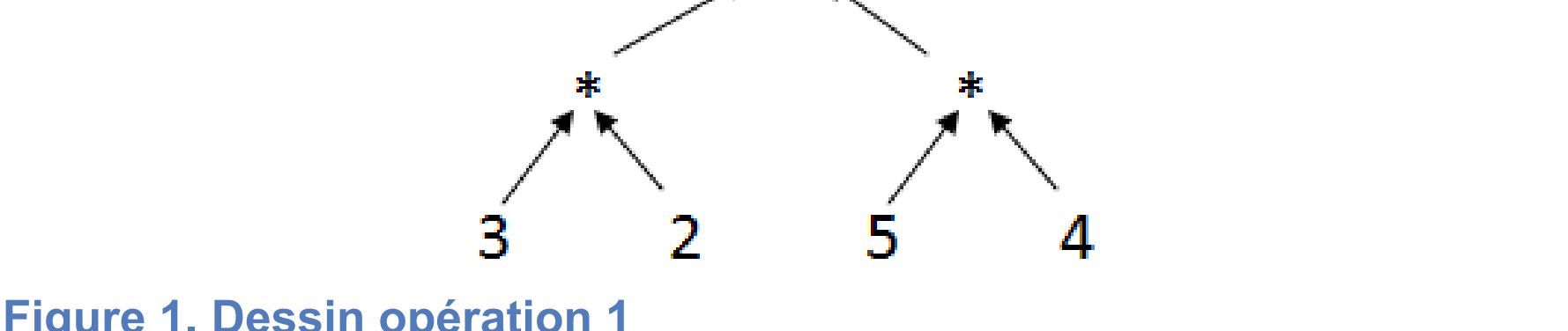


Figure 1. Dessin opération 1

Cela se traduit : **(3*2)+(5*4) = 26.**

1.3. Exemple 3 :

- Celui-ci est un peu plus complexe, mais en appliquant la méthode, on parvient au but très facilement.
- Pour cet exemple, le schéma est :

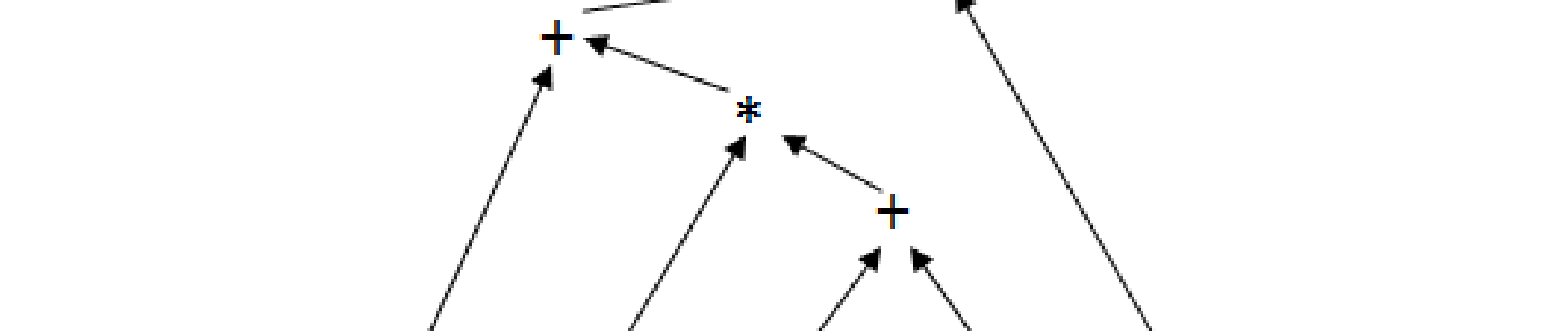


Figure 2. Dessin opération 2

Cela se traduit par : **((5+3)*2)+1)-7 = 10.**

1.4. Programme de résolution d'EPF :

Comme vous pouvez le constater, si les EPF deviennent très longues, c'est aussi très long à calculer. Nous allons donc faire le programme qui les résout en nous aidant des piles.

Principe

L'expression est lue de gauche à droite,

- . lorsqu'une valeur (opérande) est rencontrée, elle est empilée ;
- . lorsqu'un opérateur est rencontré :
 - . une première valeur est dépilée.
 - . une deuxième valeur est dépilée.
 - . on effectue l'opération correspondant à l'opérateur rencontré.
 - . on empile le résultat.
- . À la fin, il ne reste plus qu'une seule valeur dans la pile, il s'agit du résultat de l'évaluation de l'EPF.

Pour simplifier ce programme, nous prendrons plusieurs hypothèses.

Hypothèses

- . l'expression est considérée comme étant syntaxiquement correcte
- . seuls les opérateurs suivants sont considérés : + - * /
- . seuls les opérandes sont des chiffres entier (compris entre 0 et 9).

Avec ces conditions, ce n'est pas trop dur d'écrire le programme. La seule difficulté qu'il peut y avoir est de faire la différence entre un opérateur et une opérande. En effet, l'EPF est donnée par l'utilisateur sous forme de string. Nous allons donc d'abord coder deux fonctions : une pour chercher les opérateurs et l'autre pour chercher les opérandes.

2. Piles d'entiers

Modifier le **Pile.java** pour stocker des entiers plutôt que des chaînes de caractères. Créer un fichier **ProgrammeCalcullette.java** où vous copierez les fonctions présentes dans **ProgrammePile.java** en les adaptant à notre nouveau type pile (pile d'entiers et non de chaînes de caractères).

3. Exercice : ProgrammeCalcullette.java

Objectif

Nous souhaitons compléter le programme **ProgrammeCalcullette.java** afin de pouvoir l'exécuter en ligne de commande de la façon suivante :

Usage


```
> java -cp . ProgrammeCalcullette 1 2 5 3 + "*" + 7 -
```

Résultat

```
Nombre de paramètre 9

Aucune erreur détectée

Le résultat est 10
```

Remarque

On veut que le * soit interprété comme un seul caractère et non pas comme caractère spécial. Il faut pour cela l'entourer par des guillemets dans le shell, sous peine d'obtenir une exécution ressemblant à :


usage

```
> java -cp . ProgrammeCalcullette 1 2 5
3 + * + 7 -
```

Résultat

```
Nombre de paramètre 94

Erreur détectée
```

Indications

Il se peut que java ne soit pas accessible immédiatement. Remédiez-y par :

```
set path="C:\Program
Files\Java\jdk1.8.0_221\bin";%path%
```

Indications

- arguments.length donne le nombre d'éléments dans l'expression
- chaque élément sera une chaîne de longueur 1
- une chaîne sera soit un opérateur c'est-à-dire un caractère parmi + - * et /
- soit un caractère représentant un chiffre entier qu'il faudra transformer en entier en **java**

4. Pour aller plus loin

Les chaînes peuvent être des nombres entiers. Consulter la documentation de la fonction [parseInt](#), et notamment l'exception qu'elle peut soulever.