SINF 1121 - Algorithmique et structures de données - Mission 1

3 octobre 2014

Introduction

Il nous était demandé de réaliser un interpréteur de PostScript en java. Pour ce faire, nous avons dû implémenter une TAD pile ainsi qu'un ensemble de commmandes supportées par notre langage :

pstack, add, sub, mul, div, dup, exch, eq, ne, def, pop

Nous avons tout d'abord scindé notre programme en deux parties : la partie manipulation de fichier et interprétation du langage. L'avantage de cette approche est de pouvoir scinder le travail entre différents sous-groupes et d'obtenir une productivité accrue.

Manipulation de fichiers

Nous avons implémenté une classe FileAccess qui nous permet d'ouvrir et lire/écrire un fichier de données et le transformer en String utilisables par d'autres classes.
L'avantage de cette classe est la réutilisation du code, c'est-à-dire pour tout autre problème nécessitant une manipulation de fichier (lecture/écriture).

Interprétation du langage

Pour réaliser les différentes opérations, nous avons implémenté une pile à l'aide d'une liste simplement chaînée. Une telle implémentation permet de conserver les données dans le bon ordre et aussi, de les récupérer dans l'ordre adéquat.
Analysons un court exemple pour vérifier cela : **1 3 add 7 mul pstack** sera décrypté comme :

- push(1)
- push(3)

– add(pop(),pop()) (1+3), suivi d'un push(4)
– push(7)
– mul(pop(),pop() (4*7),suivi du push(28) correspondant
– pstack() et impression de la pile. (ici, uniquement 28)

Remarques d'implémentation

Nous avons utilisé une fonctionnalité de Java 8 qui nous permet de réaliser un "switch" sur des String, afin de déterminer si l'élément lu est un float ou une opération, et si c'est une opération, exécuter la fonction correspondante. L'avantage de cette solution est que si nous devons ajouter une nouvelle fonction dans notre interpréteur, il "suffit" de rajouter un cas au switch et de paramétrer la nouvelle méthode.
Imaginons que nous désirons utiliser un autre TAD, toutes nos fonctions restent valables et fonctionnelles, il ne faut que redéfinir les fonctions push et pop correspondantes au nouveau TAD (l'implémentation de l'interpréteur est indépendant de l'implémentation du TAD, et indépendant de la lecture de fichier).

Dans le cas d'une instruction non valide (par exemple **add 3 4 pstack**), nous avons utilisé le principe d'erreur. En effet, si l'interpréteur reçoit un add alors que le nombre d'éléments dans la pile est inférieur à 2 (strictement), un des deux push de la fonction add renverra une exception qui indiquera à l'utilisateur futur de notre interpréteur que ses instructions sont incorrectes. Ceci est fait au niveau du TAD et est donc indépendant de l'interpréteur. Dans le cas de changement de TAD (voir remarque ci-dessus), il ne faut pas oublier de faire envoyer une exception par la fonction pop dans le cas où la pile est vide.