le cnam
Nord-Pas de Calais

UE NFA032
Programmation Java – programmation objet

Projet TP JAVA
Mini-Calculette

Semestre N°02 2014-2015

Vincent BRABANT

UE NFA032
Programmation Java – programmation objet

1 Présentation

Il s'agit de réaliser un évaluateur d'expression arithmétique parenthésée infixe implémentant une technique d'interprétation rapide. On utilisera la notion de pile vue en cours et les classes implémentant les piles.

2 Description

Cet évaluateur sera constitué de 3 parties :
- Un analyseur lexical
- Un convertisseur d'expression
- Un automate à pile chargé de l'évaluation proprement dite de l'expression transformée.

2.1 L'analyseur lexical

L'analyseur lexical est chargé d'isoler chaque élément (parenthèse ouvrante ou fermante, opérateurs et entiers) par un parcours séquentiel de l'expression infixe. Il est réalisé sous la forme d'une classe indépendante construite autour d'un type privé. L'algorithme de cet analyseur lexical doit être explicité dans le dossier de réalisation.

2.2 Le convertisseur

Le convertisseur doit convertir une expression infixe entièrement parenthésée en expression post-fixée (voir définitions et rappels). Ces expressions sont manipulées sous forme de chaînes de caractères.

Exemple : le convertisseur transformera l'expression (5*(22+3)) en l'expression 5 22 3 + *
Pour cela, il parcourra séquentiellement l'expression en faisant appel à l'analyseur lexical et exploitera une pile. L'algorithme de ce convertisseur doit être explicité dans le dossier de réalisation.

2.3 L'automate à pile

L'automate à pile dont l'algorithme doit être explicité dans le dossier de réalisation, interprète l'expression postfixée en exploitant une pile d'entiers. Pour cela, il doit convertir les expressions représentant un nombre en l'entier correspondant. Le parcours de l'expression post-fixée se fera à l'aide de l'analyseur lexical.
L'automate à pile réalise également les opérations correspondant aux opérateurs rencontrés dans l'expression.

2.4 L'évaluateur

L'évaluateur sera développé sous la forme d'une fonction retournant en valeur le résultat de l'évaluation de l'expression infixe donnée en argument, en assurant l'enchaînement entre le convertisseur et l'automate à pile.

UE NFA032
Programmation Java – programmation objet
L'ensemble constitué du convertisseur, de l'automate à pile et de l'évaluateur sera regroupé dans une classe.

2.5 Le programme de tests

Le programme de tests exploitera les classes précédemment définies. Il sera constitué d'une séquence d'appels à l'évaluateur avec comme paramètre différentes expressions infixes judicieusement construites et montrant son bon fonctionnement.

2.6 Restrictions

On ne traitera que les 4 opérations de base :

- La multiplication notée *
- La division entière notée / (on négligera le reste de la division)
- L'addition notée +
- La soustraction notée –

Tous les opérateurs sont binaires (exactement 2 opérandes). Les entiers d'une expression seront représentés sous la forme d'un ou plusieurs chiffres (digits). Ils sont non signés (pas de chiffres négatifs)

N.B. Le résultat de l'évaluation d'une expression peut, quant à lui, avoir n'importe quelle valeur (positive ou négative).

2.6 Définitions et rappels

Une expression infixe est une expression arithmétique classique. On manipulera des expressions entièrement parenthésées pour lever toute ambiguïté et ne pas gérer les règles de priorité entre opérateurs.
Exemple : ((5+35)*3)

Une expression postfixée (polonaise inverse) est une expression où les opérandes sont placés devant l'opérateur. Une expression postfixée n'a pas besoin d'être parenthésée.
Exemple : 5 35 + 3*

3 Livraison

3.1 Dossier de réalisation

Les éléments à livrer seront groupés dans un dossier de réalisation constitué de :

• Les sources commentés et mis en forme selon les règles de programmation vues en cours et TD.
• Un dossier de conception indiquant notamment les choix de conception effectués (description des règles lexicales de l'analyseur, détection et récupération d'erreur . . .) et leur justification.
Ce dossier doit également comporter une analyse de la méthode suivie pour tester de façon unitaire les programmes et la correction apportée.
• Un dossier de tests de validation démontrant la conformité du fonctionnement du logiciel réalisé aux présentes spécifications ainsi que la couverture complète des tests.
• Une analyse du comportement du logiciel réalisé lorsqu'on exécute le programme de test fourni ci-après (voir section 6).
Le logiciel devra être structuré en classes clairement identifiées et fonctionnellement cohérentes.

3.2 Modèle algorithme

Les éléments à livrer seront groupés dans un dossier de réalisation constitué de tous les algorithmes mis sous la forme suivante :

EXERCICE :
NUMERO :
OBJET :

CREATION : jj/mm/aaaa
DEVELOP : Nom PRENOM
VERSION : 1

DATE MODIF :
MODIF :
Nelle VERSION :

Enoncé :

ETAPE 1 : Pose le problème

préconditions :

UE NFA032
Programmation Java – programmation objet
postconditions :
Objectif :

ETAPE 2 : Déclaration

ETAPE 3 : Algorithme

ETAPE 4 : Verification

3.3 Délais

Date : livraison le vendredi 19 juin 2015 à 18H30 dernier délai.

La livraison doit se faire sous la forme suivante :

- Le dossier de réalisation doit être imprimé et relié ou agrafé.
- Ce dossier doit être accompagné d'un CD // Clef USB contenant uniquement les fichiers source, les algorithmes et le dossier de réalisation sous forme informatique (format Word)
- Le tout doit être déposé ou adressé sous enveloppe au CNAM, 8 boulevard Louis XIV, 59000 Lille à Madame FLORENT, Madame DAMEZ ou Madame HAMOUR.

Il est préférable de joindre une lettre précisant le contenu de l'enveloppe, les références de l'auditeur (nom, prénom, numéro) adressée à mon intention.

4 Analyseur lexical

L'analyseur lexical est décrit par la spécification suivante :

Classe analyseur{

Type Tlexeme = (operande, operateur, parenthese_ouvrante, parenthese_fermante, separateur, inconnu)

Type Texpression (max : integer) = (
    valeur : string (1..max) ;
    position_courante : integer ;
    longueur : integer ;
    nature : Tlexeme ;
)

Procedure init(mon_expression : in out Texpression ; chaine : in chaine_de_caractères)
Procedure lexeme_suivant(mon_expression : in out Texpression)
Fonction lexeme_courant(mon_expression : Texpression) : chaine_de_caractères
Fonction nature_lexeme(mon_expression : Texpression) : Tlexeme
Fonction dans_expression(mon_expression : Texpression) : booléen

}

5 Evaluateur

L'évaluateur est décrit par la spécification suivante :

classe evaluateur {

Fonction convertisseur(infixe : chaine_de_caractères): chaine_de_caractères
Fonction automate(postfixe : chaine_de_caractères) : entier
Fonction evaluate (expression: chaine de caractère): entier;
}

6 Tests imposés

Le programme de test suivant est donné pour l'analyse du comportement de l'évaluateur réalisé sur certains cas de figure. Il ne remplacera pas le programme de test que doit réaliser l'auditeur et décrit en 2.5.

On demande d'exécuter les tests suivants et d'en commenter les résultats.

```
evaluateur.evaluate("5");
evaluateur.evaluate("(5555+3)");
evaluateur.evaluate("(8)");
evaluateur.evaluate("(3+5555)");
evaluateur.evaluate("(5*(2+3))");
evaluateur.evaluate("((125+3)*512)");
evaluateur.evaluate("(2*((((53-52)*1)/5)+(6-7)))");
evaluateur.evaluate("(2*((((3-2)*1)/(17-((2*3)+11)))+(6-7)))");
evaluateur.evaluate("(2*(( ((3-2)*1)/5)+(6-7)))");
evaluateur.evaluate("(2*(((((3-2)*1)/5)+(6-7)))");
evaluateur.evaluate("(2*((((3-2)*1)/5))+(6-7)))");
```