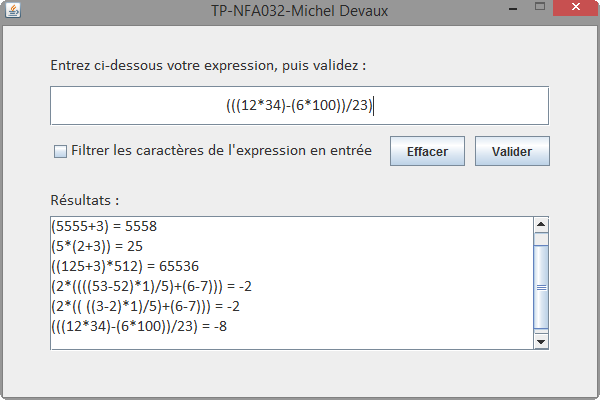
**PROJET TP JAVA  
MINI-CALCULETTE**



UE NFA032

Programmation Java – Programmation objet

**SEMESTRE N° 02 2014-2015**

TABLE DES MATIERES :

**Table des matières page 2**

**Présentation page 2  
 1. Objectif page 2**

**2. Moyens page 2**

**3. Principe de fonctionnement page 2**

**4. Les limites du programme page 3**

**Description page 3**

**1. Le type TLexeme page 3**

**2. L'Analyseur page 3**

**3. Les exceptions page 4**

**4. Structure des données page 4**

**5. Choix complémentaires page 5**

**Comportement page 6**

**1. Tests de l'énoncé page 6**

**2. Tests supplémentaires page 8**

**3. Interprétation page 8**

**4. Conformité page 9**

**Sources page 10**

**1. Algorithmes page 10**

**2. Code Java page 26**

**PRESENTATION :**

**1. Objectif :**

Concevoir un évaluateur d’expression arithmétique avec parenthèses implémentant une technique d’interprétation rapide.

**2. Moyens :**

La notion de Pile doit être utilisée ainsi que la notation post-fixée pour le calcul.

**3. Principe de fonctionnement :**

- Le programme attend la saisie d'une expression de la part de l'utilisateur.

- Cette expression est dans un premier temps analysée par un Analyseur. Les expressions incorrectes, ne pouvant être calculées, lèvent une exception.

L'Analyseur se charge également de scinder l'expression en éléments primaires de nature : opérande, opérateur, parenthèse ouverte ou parenthèse fermée.   
Ces éléments sont ordonnés en une suite infixée.

- La suite infixée produite par l'Analyseur est passée à un Convertisseur qui se charge de la transformer en une suite post-fixée de type notation polonaise inverse. Les éléments primaires, leur valeur et leur nature, sont préservés.

Le Convertisseur utilise une Pile.

- Un Automate se charge de calculer le résultat de la suite post-fixée. Cet automate utilise lui-aussi une Pile.  
Les erreurs de calcul, comme la division par zéro ou le dépassement de capacité, entraînent la levée d'une exception.

- Une unité de Traitement coordonne l'Analyseur, le Convertisseur et l'Automate.   
Cette unité reçoit en entrée l'expression saisie par l'utilisateur et renvoie en sortie le résultat sous forme d'entier.   
Les exceptions produites par l'Analyseur et l'Automate sont renvoyées à l'utilisateur.

**4. Les limites du programme :**

- Le programme ne traite que les entiers.

- Seules les quatre opérations primaires – addition, soustraction, multiplication et division – sont autorisées.

- Les opérateurs sont obligatoirement binaires – avec seulement deux opérandes – et sont délimités par des parenthèses.

**DESCRIPTION :**

**1. Le type TLexeme :**

Le type TLexeme est une classe représentant un élément primaire de l'expression mathématique à calculer.  
Il possède une énumération et deux attributs.

L'énumération (TNature) :   
Elle reprend les types utilisables dans l'expression : opérande, opérateur, parenthèse ouverte et parenthèse fermée.

Les attributs :

Valeur : de type chaîne, il contient la valeur de l'élément.

Nature : de type TNature, il contient la nature de l'élément.

**2. L'Analyseur :**

Le rôle de l'Analyseur est double : établir la liste des lexèmes et analyser la structure de l'expression.  
  
Etablir la liste des lexèmes :  
Chaque caractère de l'expression est analysé selon qu'il s'agit d'un opérateur, d'un chiffre, d'une parenthèse, d'un espace ou de tout autre caractère non défini.  
Les chiffres sont regroupés pour former des nombres. Les espaces sont omis. Les caractères non admis déclenchent une exception.  
La liste des lexèmes est placée dans une File typée. Les lexèmes peuvent ainsi être rappelés dans l'ordre dans lequel ils sont entrés (principe FIFO).

Analyser la structure de l'expression :

On part du principe que la structure infixée de l'expression saisie, aussi complexe qu'elle soit, est une composition d'expressions primaires de forme (**n**•**n**) où **n** est un opérande et • un opérateur.

L'analyse lexicale consiste à effectuer une boucle unique chargée de remplacer progressivement chaque expression (**n**•**n**) par **n**.  
Le résultat final doit, dans le cas d'une expression correctement formulée, être égal à **n**.

Si tel n'est pas le résultat – sous réserve de tolérances particulières – la structure est incorrecte et une exception est levée.

Exemple :

L'expression (((12 + 2) x (53 - 16)) + 21) a pour structure (((n•n) •(n•n)) •n)

La réduction progressive de cette structure s'effectue ainsi :

((n•n)•(n•n))•n) ⭢ *Réduction des deux expressions soulignées*

((n•n)•n) ⭢ *Réduction de l'expression soulignée*

(n•n) ⭢ *Réduction de l'expression soulignée*

n ⭢ *Résultat final*

**3. Les exceptions :**

Les exceptions sont de deux natures :

- Exceptions levées par des erreurs de syntaxe : erreur de structure, parenthèse, opérateur ou opérande manquant ou excédentaire.

- Exceptions levées par des erreurs arithmétiques : division par zéro, dépassement de capacité.

Les exceptions sont générées par les méthodes de bas niveaux – méthodes de l'Analyseur, l'Automate – et sont progressivement remontées jusque l'interface utilisateur qui les traite.  
La gestion des erreurs s'en trouve réduite et centralisée.

**4. Structure des données :**

L'architecture des données se base sur trois classes :

- La classe TLexeme qui est décrite en page 2.   
Les instances de cette classe sont initialement générées par l'Analyseur. Ces objets sont ensuite réordonnés par le Convertisseur, puis utilisés par l'Automate pour calculer le résultat de l'expression.

- La classe File.

Cette classe représente une liste d'éléments génériques chaînés organisés selon le principe FIFO (First In First Out).  
L'Analyseur, le Convertisseur et l'Automate utilisent une instance de la classe File pour organiser les lexèmes dans un ordre bien précis d'entrée/sortie.

- La classe Pile.

Cette classe représente une liste d'éléments génériques chaînés organisés selon le principe LIFO (Last In First Out).  
Le Convertisseur utilise une instance de cette classe Pile pour réorganiser les lexèmes, issus de la File produite par l'Analyseur, en une nouvelle File de type post-fixé.  
L'Automate utilise lui aussi une Pile pour effectuer dans le bon ordre le calcul de l'expression post-fixée représentée par la File issue du Convertisseur.

**5. Choix complémentaires :**

Le programme est organisé en trois packages selon le patron de conception Modèle-Vue-Contrôleur.

Package tp.modele :

Ce package regroupe les classes représentant les données et les classes représentant les exceptions :

- TLexeme

- File

- Pile

- ExceptionMath

- ExceptionStruct

Package tp.controleur :

Ce package regroupe les classes dédiées au traitement de l'expression et des données :

- Analyseur

- Evaluateur

- IEvaluateur

- MainTP

La classe IEvaluateur est l'interface de la classe Evaluateur qui se charge des différentes étapes menant de l'expression saisie au renvoi de son résultat.

La classe MainTP contient la méthode main qui lance l'interface utilisateur.

Package tp.vue :

Ce package regroupe les classes dédiées à l'interface utilisateur :

- IHMConsole

- IHMGraphique

- TextFieldFiltre

- TextAreaScrollable

Deux types d'interface sont disponibles : une interface console et une interface graphique utilisant Swing.

Le développeur choisit l'interface à lancer en paramétrant la variable (booléen) modeGraphique de MainTP.

L'interface graphique utilise deux composants spécifiquement développés pour le programme :

- TextFieldFiltre : est un composant hérité de JTextField qui permet de filtrer les caractères en entrée.

Si l'utilisateur coche la case "Filtrer les caractères en entrée", les seuls caractères admissibles sont ceux acceptés par l'Analyseur.  
Ce composant valide en outre la touche "Enter" comme action de déclenchement du traitement de l'expression : l'utilisateur entre son expression, appuie sur "Enter" et l'expression est calculée.

- TextAreaScrollable : est un composant hérité de JPanel qui combine un composant JTextArea et un composant JScrollPane. Il permet d'obtenir une zone texte associée à une barre de défilement.  
Ce composant est utilisé pour stocker les expressions calculées.

**COMPORTEMENT :**

Pour ses tests, le programme utilise JUnit.   
Une classe EvaluateurTest regroupe les tests définis dans l'énoncé et les tests supplémentaires.

**1. Tests de l'énoncé :**

@Test(timeout = 1000)

**public** **void** testCalculer01()

**throws** ExceptionStruct, ExceptionMath {

// assetEquals(valeur attendue, valeur calculée, tolérance)

Assert.*assertEquals*(5, Evaluateur.*getEvaluateur*()

.traitement("5"), 0);

}

@Test(timeout = 1000)

**public** **void** testCalculer02()

**throws** ExceptionStruct, ExceptionMath {

Assert.*assertEquals*(5558, Evaluateur.*getEvaluateur*()

.traitement("(5555+3)"), 0);

}

@Test(timeout = 1000)

**public** **void** testCalculer03()

**throws** ExceptionStruct, ExceptionMath {

Assert.*assertEquals*(8, Evaluateur.*getEvaluateur*()

.traitement("(8)"), 0);

}

@Test(timeout = 1000)

**public** **void** testCalculer04()

**throws** ExceptionStruct, ExceptionMath {

Assert.*assertEquals*(5558,Evaluateur.*getEvaluateur*()

.traitement("(3+5555)"), 0);

}

@Test(timeout = 1000)

**public** **void** testCalculer05()

**throws** ExceptionStruct, ExceptionMath {

Assert.*assertEquals*(25, Evaluateur.*getEvaluateur*()

.traitement("(5\*(2+3))"), 0);

}

@Test(timeout = 1000)

**public** **void** testCalculer06()

**throws** ExceptionStruct, ExceptionMath {

Assert.*assertEquals*(65536, Evaluateur.*getEvaluateur*()

.traitement("((125+3)\*512)"), 0);

}

@Test(timeout = 1000)

**public** **void** testCalculer07()

**throws** ExceptionStruct, ExceptionMath {

Assert.*assertEquals*(-2, Evaluateur.*getEvaluateur*()

.traitement("(2\*((((53-52)\*1)/5)+(6-7)))"), 0);

}

@Test(expected = ExceptionMath.**class**, timeout = 1000)

**public** **void** testCalculer08()

**throws** ExceptionStruct, ExceptionMath {

**int** res = Evaluateur.*getEvaluateur*()

.traitement("(2\*((((3-2)\*1)/(17-((2\*3)+11)))+(6-7)))");

}

@Test(timeout = 1000)

**public** **void** testCalculer09()

**throws** ExceptionStruct, ExceptionMath {

Assert.*assertEquals*(-2,Evaluateur.*getEvaluateur*()

.traitement("(2\*(( ((3-2)\*1)/5)+(6-7)))"), 0);

}

@Test(expected = ExceptionStruct.**class**, timeout = 1000)

**public** **void** testCalculer10()

**throws** ExceptionStruct, ExceptionMath {

**int** res = Evaluateur.*getEvaluateur*()

.traitement("(2\*(((((3-2)\*1)/5)+(6-7)))");

}

@Test(expected = ExceptionStruct.**class**, timeout = 1000)

**public** **void** testCalculer11()

**throws** ExceptionStruct, ExceptionMath {

**int** res = Evaluateur.*getEvaluateur*()

.traitement("(2\*((((3-2)\*1)/5))+(6-7))))");

}

**2. Tests supplémentaires :**

@Test(expected = ExceptionStruct.**class**, timeout = 1000)

**public** **void** testCalculer12()

**throws** ExceptionStruct, ExceptionMath {

**int** res = Evaluateur.*getEvaluateur*()

.traitement("()");

}

@Test(expected = ExceptionStruct.**class**, timeout = 1000)

**public** **void** testCalculer13()

**throws** ExceptionStruct, ExceptionMath {

**int** res = Evaluateur.*getEvaluateur*()

.traitement("(5++3");

}

@Test(expected = ExceptionStruct.**class**, timeout = 1000)

**public** **void** testCalculer14()

**throws** ExceptionStruct, ExceptionMath {

**int** res = Evaluateur.*getEvaluateur*()

.traitement("(-53)");

}

@Test(timeout = 1000)

**public** **void** testCalculer15()

**throws** ExceptionStruct, ExceptionMath {

Assert.*assertEquals*(523, Evaluateur.*getEvaluateur*()

.traitement("((523))"), 0);

}

@Test(expected = ExceptionMath.**class**, timeout = 1000)

**public** **void** testCalculer16()

**throws** ExceptionStruct, ExceptionMath {

**int** res = Evaluateur.*getEvaluateur*()

.traitement("(99999999999 + 2)");

}

@Test(expected = ExceptionMath.**class**, timeout = 1000)

**public** **void** testCalculer17()

**throws** ExceptionStruct, ExceptionMath {

**int** res = Evaluateur.*getEvaluateur*()

.traitement("((123456\*123456)\*123456)");

}

**3. Interprétation :**

Si les résultats des tests valident globalement le programme et son fonctionnement, ils ont leurs limites, notamment celle qui concerne le dépassement de capacité du type de donnée int.  
  
L'énoncé impose un calcul sur des entiers.

Mais que se passe t-il lorsque le résultat du calcul déborde les capacités de stockage d'un entier ?

Contrairement à toute attente, la machine Java ne lève pas d'erreur mais renvoie un résultat, résultat bien évidemment faux.

Ainsi au calcul du carré de l'entier 123456, Java renvoie -1938485248 sans indiquer d'erreur alors que le résultat est 15241383936.

**int** nb = 123456;

**int** calculEntier = nb \* nb;

**long** calculLong = (**long**)nb \* (**long**)nb;

System.***out***.println(calculEntier + " " + calculLong);

* -1938485248 15241383936

Malgré cela, le programme intercepte les dépassements de capacité même lorsqu'ils ne sont pas signalés par Java.  
Dans l'automate, au niveau du calcul, un test vérifie pour chaque opération s'il n'y a pas de débordement. Dans l'affirmative, une exception est levée.

**4. Conformité :**

Le programme répond au cahier des charges :

- Il est capable d'évaluer une expression arithmétique avec parenthèses selon les critères initialement définis.

- L'analyse de la structure est à la fois simple, très rapide et n'a pas montré de failles lors des différents tests (tests exigés et tests complémentaires).

- Comme demandé, le programme utilise une Pile et un type primaire (TLexeme) ayant pour attribut la nature et la valeur de l'objet.

- Les erreurs sont interceptées et signalées à l'utilisateur. Ces erreurs n'entraînent pas l'interruption brutale du programme.

- Le programme peut optionnellement faire preuve de tolérance au niveau de la syntaxe pour peu que cette dernière soit cohérente.

- Le programme implante une interface utilisateur sûre et intuitive.

- L'Evaluateur, son Interface, et les classes qui lui sont associées pourraient aisément être réutilisés dans un autre type de programme.

**SOURCES :**

**1. Algorithmes :**

**TLexeme**

OBJET : Définit un type représentant la valeur et la nature d'un élément d'une expression arithmétique

CREATION : 23/04/2015

MODIFIE :

VERSION : 1.0

DEVELOP : Michel Devaux

TYPE TNature = (

operande,

operateur,

parenthese\_

ouvrante,

parenthese\_fermante

)

Type TLexeme = (

valeur : Chaine

nature : TNature

)

// PROCEDURE : CREER\_TLEXEME

// OBJET : Créer un nouvel objet de type TLexeme  
// PRECONDITION :   
// POSTCONDITION : Retourne un tableau objet de type TLexeme.

PROCEDURE CREER\_TLEXEME (OUT uneExpression : TLexeme)

FAIRE

valeur (uneExpression) := NUL

nature (uneExpression) := NUL

FIN\_FAIRE

**La Pile**

OBJET : Mémoriser les données de type TLexeme dans un type abstrait ou seul le dernier élément entré peut être lu et où le premier élément à sortir est le dernier entré (LIFO)

CREATION : 23/04/2015

MODIFIE :

VERSION : 1.0

DEVELOP : Michel Devaux

Type TElement (

valeur : Entier

suivant : TElement

)

Type Pile (

tete : TElement

)

// PROCEDURE : CREER\_PILE

// OBJET : Créer un nouvel objet de type Pile  
// PRECONDITION :   
// POSTCONDITION : Retourne un objet Pile.

PROCEDURE CREER\_PILE (OUT laPile : Pile)

FAIRE

tete(laPile) := NUL

FIN\_FAIRE

// FONCTION : PILE\_VIDE

// OBJET : Savoir si la Pile est vide ou non  
// PRECONDITION :   
// POSTCONDITION : Retourne VRAI si la Pile est vide

FONCTION PILE\_VIDE (laPile : Pile) : Booleen

FAIRE

rtn : Booleen

SI tete (laPile) = NUL

rtn := VRAI

SINON

rtn := FAUX

FIN\_SI

PILE\_VIDE := rtn

FIN FAIRE

// FONCTION : LIRE\_PILE

// OBJET : Lire le dernier élément empilé  
// PRECONDITION :   
// POSTCONDITION : Retourne un objet de type TLexeme

FONCTION LIRE\_PILE (laPile : Pile) : TLexeme

FAIRE

rtn : TLexeme

SI PILE\_VIDE (laPile) = VRAI

rtn := NUL

SINON

rtn := valeur( CONTENU(tete (laPile)))

FIN\_SI

LIRE\_PILE := rtn

FIN FAIRE

// PROCEDURE : EMPILER

// OBJET : Empiler un élément  
// PRECONDITION : Reçoit un élément de type TLexeme  
// POSTCONDITION : Lève une Exception si la Pile est pleine

PROCEDURE EMPILER (IN OUT laPile : Pile, IN exp : TLexeme)

FAIRE

element : TElement

element := CREER\_ELEMENT

valeur (CONTENU (element)) := exp

suivant (CONTENU (element)) := tete (laPile)

tete (laPile) := element

FIN FAIRE

// PROCEDURE : DEPILER

// OBJET : Retire de la Pile le dernier élément entré  
// PRECONDITION :   
// POSTCONDITION :

PROCEDURE DEPILER (IN OUT laPile : Pile)

FAIRE

SI PILE\_VIDE (laPile) = FAUX

element : TElement

element := tete (laPile)

tete (laPile) := suivant (CONTENU (tete (laFile)))

element := NUL

FIN\_SI

FIN FAIRE

**La File**

OBJET : Mémoriser les données de type TLexeme dans un type abstrait ou seul le premier élément entré peut être lu et où le premier élément à sortir et le premier entré (LIFO)

CREATION : 23/04/2015

MODIFIE :

VERSION : 1.0

DEVELOP : Michel Devaux

Type TElement (

valeur : Entier

suivant : TElement

)

Type File (

tete : TElement

queue : TElement

)

// PROCEDURE : CREER\_FILE

// OBJET : Créer un nouvel objet de type File  
// PRECONDITION :   
// POSTCONDITION : Retourne un tableau objet de type File

PROCEDURE CREER\_FILE (OUT laFile : File)

FAIRE

tete (laFile) := NUL

queue (laFile) := NUL

FIN\_FAIRE

// FONCTION : FILE\_VIDE

// OBJET : Savoir si la File est vide ou non  
// PRECONDITION :   
// POSTCONDITION : Retourne VRAI si la File est vide

FONCTION FILE\_VIDE (laFile : File) : Booleen

FAIRE

rtn : Booleen

SI tete (laFile) = NUL

rtn := VRAI

SINON

rtn := FAUX

FIN\_SI

FILE\_VIDE := rtn

FIN FAIRE

// FONCTION : LIRE\_FILE

// OBJET : Lire le premier objet de la File  
// PRECONDITION :   
// POSTCONDITION : Retourne un objet de type TLexeme

FONCTION LIRE\_FILE (laFile : File) : TLexeme

FAIRE

exp := TLexeme

SI FILE\_VIDE (laFile) = FAUX

exp := valeur( CONTENU(tete (laFile)))

SINON

exp := NUL

FIN\_SI

LIRE\_FILE := exp

FIN FAIRE

// PROCEDURE : ENFILER

// OBJET : Ajoute un élément à la File  
// PRECONDITION : Reçoit un élément de type TLexeme  
// POSTCONDITION :

PROCEDURE ENFILER (IN OUT laFile : File, IN exp : TLexeme)

FAIRE

element : TElement

element := CREER\_ELEMENT

valeur (CONTENU (element)) := exp

suivant (CONTENU (element)) := NUL

SI FILE\_VIDE (laFile) = VRAI

tete (laFile) := element

queue (laFile) := element

SINON

suivant (CONTENU (queue (laFile))) := element

queue (laFile) := element

FIN\_SI

FIN\_FAIRE

// PROCEDURE : DEFILER

// OBJET : Retire un élément à la File  
// PRECONDITION :   
// POSTCONDITION :

PROCEDURE ENFILER (IN OUT laFile : File)

FAIRE

SI FILE\_VIDE (laFile) = FAUX

element : TElement

element := tete (laFile)

tete (laFile) = suivant (CONTENU (tete (laFile)))

element := NUL

FIN\_SI

FIN\_FAIRE

**FONCTIONS Chaines**

OBJET : Représente un ensemble de fonctions destinées à la manipulation des chaînes de caractères

CREATION : 23/04/2015

MODIFIE :

VERSION : 1.0

DEVELOP : Michel Devaux

// FONCTION : SOUS\_CHAINE

// OBJET : Obtenir une portion de la chaine initiale.  
// PRECONDITION : Reçoit une chaîne  
// POSTCONDITION : Retourne une chaîne

FONCTION SOUS\_CHAINE (IN texte : Chaine, IN debut : Entier, IN longueur : Entier) : Chaine

FAIRE

rtn : Chaine

i : Entier

//

rtn := ""

SI LONGUEUR (texte) >= longueur

POUR i := debut JUSQUE debut + longueur - 1

rtn = rtn + texte[i]

i\_SUIVANT

FIN\_SI

SOUS\_CHAINE := rtn

FIN\_FAIRE

// FONCTION : COMPARER\_CHAINES

// OBJET : Définir si 2 chaînes sont identiques  
// PRECONDITION : Reçoit deux chaînes  
// POSTCONDITION : Retourne un booleen

FONCTION COMPARER\_CHAINES (IN chaine1 : Chaine, IN chaine2 : Chaine) : Booleen

FAIRE

rtn : booleen

i : Entier

//

rtn := VRAI

SI LONGUEUR (chaine1) != LONGUEUR (chaine2)

rtn := FAUX

SINON

POUR i := 0 JUSQUE LONGUEUR (chaine1) - 1

SI chaine1[i] != chaine2[i]

rtn := FAUX

FIN\_SI

i\_SUIVANT

FIN\_SI

COMPARER\_CHAINES := rtn

FIN\_FAIRE

// FONCTION : POSITION\_DANS

// OBJET : Obtenir la position de la 1ère occurrence d'une sous-chaîne dans une autre  
// PRECONDITION : Reçoit deux chaînes  
// POSTCONDITION : Retourne un entier

FONCTION POSITION\_DANS (IN texte : Chaine, IN aChercher : Chaine) : Entier

FAIRE

position, i : Entier

//

position := -1

SI LONGUEUR (aChercher) <= LONGUEUR (texte)

POUR i := 0 JUSQUE LONGUEUR (texte) - LONGUEUR (aChercher)

sousChaine = SOUS\_CHAINE (texte, i, LONGUEUR (aChercher))

SI POSITION = -1 ET COMPARER\_CHAINES (sousChaine, aChercher) = VRAI

position := i

FIN\_SI

i\_SUIVANT

FIN\_SI

POSITION\_DANS := position

FIN\_FAIRE

// FONCTION : CONTENU\_DANS

// OBJET : Définir si une chaîne est incluse dans une autre  
// PRECONDITION : Reçoit deux chaînes  
// POSTCONDITION : Retourne un booleen

FONCTION CONTENU\_DANS (IN texte : Chaine, IN aChercher : Chaine) : Booleen

FAIRE

rtn : Booleen

i : Entier

//

rtn := FAUX

SI LONGUEUR (aChercher) <= LONGUEUR (texte)

POUR i := 0 JUSQUE LONGUEUR (texte) - LONGUEUR (aChercher)

sousChaine = SOUS\_CHAINE (texte, i, LONGUEUR (aChercher))

SI rtn = FAUX ET COMPARER\_CHAINES (sousChaine, aChercher) = VRAI

rtn := VRAI

FIN\_SI

i\_SUIVANT

FIN\_SI

CONTENU\_DANS := rtn

FIN\_FAIRE

// FONCTION : REMPLACER\_CHAINE

// OBJET : Remplacer une portion de chaîne par une autre à l'intérieur d'une chaîne  
// PRECONDITION : Reçoit trois chaînes  
// POSTCONDITION : Retourne une chaîne

FONCTION REMPLACER\_CHAINE (IN texte : Chaine, IN aChercher : Chaine, IN aRemplacer : Chaine) : Chaine

FAIRE

rtn, temp : Chaine

position, debut : Entier

//

rtn := texte

TANT\_QUE POSITION\_DANS (rtn, aChercher) != -1

temp := ""

position := POSITION\_DANS (rtn, aChercher)

SI position > 0

temp := SOUS\_CHAINE (rtn, 0, position)

FIN\_SI

temp := temp + aRemplacer

SI position < LONGUEUR (rtn) - LONGUEUR (aChercher)

debut := position + LONGUEUR (aChercher)

temp := temp + SOUS\_CHAINE (rtn, debut, LONGUEUR (rtn) – debut)

FIN\_SI

rtn := temp

FIN\_TANT\_QUE

REMPLACER\_CHAINE := rtn

FIN\_FAIRE

**Analyseur**

OBJET : Convertit une expression arithmétique en une File d'éléments de type TLexeme

CREATION : 23/04/2015

MODIFIE :

VERSION : 1.0

DEVELOP : Michel Devaux

// Constantes

CHIFFRES = "0123456789"

SIGNES = "+-\*/"

PARENTHESES = "()"

fileLexemes : File

// PROCEDURE : CREER\_ANALYSEUR

// OBJET : Créer un nouvel objet de type Analyseur  
// PRECONDITION :   
// POSTCONDITION : Retourne un tableau objet de type Analyseur.

PROCEDURE CREER\_ANALYSEUR (OUT anl : Analyseur)

FAIRE

fileLexemes := NUL

FIN\_FAIRE

// FONCTION : LISTER\_EXPRESSIONS

// OBJET : Obtient, en les analysant, les éléments qui composent l'expression et les range dans un tableau.  
// PRECONDITION : Reçoit une chaîne  
// POSTCONDITION : Retourne un tableau composé d'éléments de type TLexeme.

// EXCEPTION : Retourne une exception si l'expression contient un caractère non autorisé ou si sa structure   
// n'est pas correcte

FONCTION LISTER\_EXPRESSIONS (IN OUT anl : Analyseur, IN expression : Chaine) : File

FAIRE

i : Entier

str, structure : Chaine

exp : TLexeme

//

// Initialisation de la File

fileLexemes (anl) : = CREER \_FILE

//

// 1ere partie : Inventorier et stocker les éléments

i := 0

structure := ""

TANT\_QUE i < LONGUEUR (expression)

str := SOUS\_CHAINE (expression, i, 1)

//

SI CONTENU\_DANS (CHIFFRE, str) = VRAI

// Traitement des chiffres : plusieurs chiffres constituent un nombre

TANT\_QUE i + 1 < LONGUEUR (expression) ET

CONTENU\_DANS (CHIFFRE, SOUS\_CHAINE (expression, i + 1, 1)) = VRAI

i := i + 1

str := str + SOUS\_CHAINE (expression, i , 1)

FIN\_TANT\_QUE

// Ajouter le nombre à la File

exp := CREER\_TLEXEME

valeur (CONTENU (exp)) := str

nature (CONTENU (exp)) := operande

enfiler(fileLexemes (anl), exp)

// Ajouter un nombre à la structure

structure := structure + "n"

SINON\_SI CONTENU\_DANS (SIGNES, str) = VRAI

// Ajouter le signe à la File

exp := CREER\_TLEXEME

valeur (CONTENU (exp)) := str

nature (CONTENU (exp)) := operateur

enfiler(fileLexemes (anl), exp)

// Ajouter un signe à la structure

structure := structure + "."

SINON\_SI CONTENU\_DANS (PARENTHESES, str) = VRAI

// Ajouter la parenthèse à la File

exp := CREER\_TLEXEME

valeur (CONTENU (exp)) := str

SI COMPARER\_CHAINE("(", str) = VRAI

nature (CONTENU (exp)) := parenthese\_ouvrante

SINON

nature (CONTENU (exp)) := parenthese\_fermante

FIN\_SI

enfiler(fileLexemes (anl), exp)

// Ajouter une parenthèse à la structure

structure := structure + str

SINON\_SI COMPARER\_CHAINES(" ", str) = VRAI

// On ne met pas les espaces dans le tableau mais ils ne provoquent pas d'erreur

SINON

// Caractère indésirable. On envoie une Exception

LEVER\_EXCEPTION ("L'expression comporte des caractères inadéquats.")

FIN\_SI

i := i +1

FIN\_TANT\_QUE

//

// 2eme partie : Evaluation de la structure  
 aChercher, aRemplacer : Chaine

aChercher := "(n.n)"

aRemplacer := "n"

TANT\_QUE CONTENU\_DANS (structure, aChercher) = VRAI

structure := REMPLACER\_DANS (structure, aChercher, aRemplacer)

FIN\_TANT\_QUE

SI COMPARER\_CHAINES (structure, aRemplacer) = FAUX

LEVER\_EXCEPTION ("La structure de l'expression n'est pas correcte.")

FIN\_SI

//

LISTER\_EXPRESSIONS := fileLexemes (anl)

FIN FAIRE

**CLASSE : Evaluateur**

OBJET : Permettre l'analyse, la transformation en expression post-fixée et le calcul d'une expression arithmétique.

CREATION : 23/04/2015

MODIFIE :

VERSION : 1.0

DEVELOP : Michel Devaux

// FONCTION : ANALYSER

// OBJET : Obtient, à partir d'une instance de l'Analyseur, une File d'éléments de type TLexeme

// PRECONDITION : Reçoit une chaîne  
// POSTCONDITION : Retourne une File composée d'éléments de type TLexeme.

// EXCEPTION : Répercute une exception éventuellement levée par Analyseur

FONCTION ANALYSER (expression : Chaine) : File

FAIRE

anl : Analyseur

fileInfixe : File

//

anl := CREER\_ANALYSEUR

fileInfixe := LISTER\_EXPRESSIONS (anl, expression)

ANALYSER := fileInfixe

FIN\_FAIRE

// FONCTION : CONVERTIR

// OBJET : Convertit une expression infixée en expression postfixée.

// PRECONDITION : Reçoit une File composée d'éléments de type TLexeme,.

// POSTCONDITION : Retourne une File composée d'éléments de type TLexeme.

FONCTION CONVERTIR (infixe : File) : File

FAIRE

postFixe : File

maPile : Pile

exp : TLexeme

//

postfixe := CREER\_FILE

maPile := CREER\_PILE

TANT\_QUE FILE\_VIDE (infixe) = FAUX

exp := LIRE\_FILE (infixe)

DEFILER (infixe)

SI nature (CONTENU (exp)) = operande

ENFILER (postfixe, exp)

SINON\_SI nature (CONTENU (exp)) = operateur

EMPILER (maPile, exp)

SINON\_SI nature (CONTENU (exp)) = parenthese\_ouvrante

EMPILER (maPile, exp)

SINON

// Parenthèse fermante

Exp := LIRE\_PILE (infixe)

TANT\_QUE nature (CONTENU (exp)) != parenthese\_ouvrante

ENFILER (postfixe, exp)

DEPILER (maPile)

exp := LIRE\_PILE (maPile)

FIN\_TANT\_QUE

DEPILER (maPile)

FIN\_SI

FIN\_TANT\_QUE

TANT\_QUE PILE\_VIDE (maPile) = FAUX

exp := LIRE\_PILE (maPile)

ENFILER (postfixe, exp)

DEPILER (maPile)

FIN\_TANT\_QUE

CONVERTIR := postfixe

FIN\_FAIRE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TEST DE LA FONCTION CONVERTIR AVEC : ((5+2)\*3)** | | | | |
| **Action** | **exp : TLexeme** | **infixe : File** | **postfixe : File** | **maPile : Pile** |
| Lire + Défiler infixe | ( | (5+2)\*3) |  |  |
| Empiler maPile | ( | (5+2)\*3) |  | ( |
| Lire + Défiler infixe | ( | 5+2)\*3) |  | ( |
| Empiler maPile | ( | 5+2)\*3) |  | (( |
| Lire + Défiler infixe | 5 | +2)\*3) |  | (( |
| Enfiler postfixe | 5 | +2)\*3) | 5 | (( |
| Lire + Défiler infixe | + | 2)\*3) | 5 | (( |
| Empiler maPile | + | 2)\*3) | 5 | ((+ |
| Lire + Défiler infixe | 2 | )\*3) | 5 | ((+ |
| Enfiler postfixe | 2 | )\*3) | 52 | ((+ |
| Lire + Défiler infixe | ) | \*3) | 52 | ((+ |
| Lire + Dépiler maPile | + | \*3) | 52 | (( |
| Enfiler postFixe | + | \*3 | 52+ | (( |
| Lire + Dépiler maPile | ( | \*3) | 52+ | ( |
| Lire + Défiler infixe | \* | 3) | 52+ | ( |
| Empiler maPile | \* | 3) | 52+ | (\* |
| Lire + Défiler infixe | 3 | ) | 52+ | (\* |
| Enfiler postfixe | 3 | ) | 52+3 | (\* |
| Lire + Défiler infixe | ) |  | 52+3 | (\* |
| Lire + Dépiler maPile | \* |  | 52+3 | ( |
| Enfiler postFixe | \* |  | 52+3\* | ( |
| Lire + Dépiler maPile | ( |  | 52+3\* |  |
| **Retourner posfixe** | **52+3\*** | | | |

// FONCTION : AUTOMATE

// OBJET : Retourne le résultat du calcul d'une expression postfixée passée en paramètre

// PRECONDITION : Reçoit une File composée d'éléments de type TLexeme.

// POSTCONDITION : Retourne un entier

// EXCEPTION : Répercute une exception éventuellement levée par la Pile

FONCTION AUTOMATE (postfixe : File) : Entier

FAIRE

maPile : Pile

exp : TLexeme

val1, val2 : Entier

//

maPile := CREER\_PILE

TANT\_QUE FILE\_VIDE (postfixe) = FAUX

exp := LIRE\_FILE (postfixe)

DEFILER (postfixe)

SI nature(CONTENU(exp)) = operande

EMPILER (maPile, exp)

SINON

val2 := valeur (CONTENU (LIRE\_PILE (maPile))

DEPILER (maPile)

val1 := valeur (CONTENU (LIRE\_PILE (maPile))

DEPILER (maPile)

SI COMPARER\_CHAINES (valeur (CONTENU (exp)), "+") = VRAI

val1 := val1 + val2

SINON\_SI COMPARER\_CHAINES (valeur (CONTENU (exp )), "-") = VRAI val1 := val1 - val2

SINON\_SI COMPARER\_CHAINES (valeur (CONTENU (exp)), "\*") = VRAI val1 := val1 \* val2

SINON

SI val2 != 0

val1 := val1 / val2

SINON

LEVER\_EXCEPTION ("Division par zéro.")

FIN\_SI

FIN\_SI

exp := ALLOUER TLexeme

valeur (CONTENU (exp)) := val1

nature (CONTENU (exp)) := operande

EMPILER (maPile, exp)

FIN\_SI

FIN\_TANT\_QUE

//

val1 := valeur ( CONTENU (LIRE\_PILE (maPile))

DEPILER (maPile)

AUTOMATE := val1

FIN\_FAIRE

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **TEST DE LA FONCTION AUTOMATE AVEC : 52+3\*** | | | | | | |
| **Action** | **exp : TLexeme** | **postfixe : File** | **maPile :**  **Pile** | **Val1 : Entier** | **Val2 : Entier** | **Val3 : Entier** |
| Lire + Défiler postfixe | 5 | 2+3\* |  |  |  |  |
| Empiler maPile | 5 | 2+3\* | 5 |  |  |  |
| Lire + Défiler postfixe | 2 | +3\* | 5 |  |  |  |
| Empiler maPile | 2 | +3\* | 52 |  |  |  |
| Lire + Défiler postfixe | + | 3\* | 52 |  |  |  |
| Val2 = Lire + Dépiler maPile | + | 3\* | 5 |  | 2 |  |
| Val1 = Lire + Dépiler maPile | + | 3\* |  | 5 | 2 |  |
| Val3 = Calculer Val1 et Val2 | + | 3\* |  | 5 | 2 | 7 |
| Créer Exp avec Val3 | 7 | 3\* |  | 5 | 2 | 7 |
| Empiler maPile | 7 | 3\* | 7 | 5 | 2 | 7 |
| Lire + Défiler postfixe | 3 | \* | 7 | 5 | 2 | 7 |
| Empiler maPile | 3 | \* | 73 | 5 | 2 | 7 |
| Lire + Défiler postfixe | \* |  | 73 | 5 | 2 | 7 |
| Val2 = Lire + Dépiler maPile | \* |  | 7 | 5 | 3 | 7 |
| Val1 = Lire + Dépiler maPile | \* |  |  | 7 | 3 | 7 |
| Val3 = Calculer Val1 et Val2 | \* |  |  | 7 | 3 | 21 |
| Créer Exp avec Val3 | 21 |  |  | 7 | 3 | 21 |
| Empiler maPile | 21 |  | 21 | 7 | 3 | 21 |
| Lire + Depiler maPile | 21 |  |  | 7 | 3 | 21 |
| **Retourner valeur Exp** | **21** | | | | | |

// FONCTION : CALCULER

// OBJET : Retourne le résultat du calcul d'une expression arithmétique passée en paramètre…  
// …Utilise successivement les 3 précédentes méthodes de l'Evaluateur

// PRECONDITION : Reçoit une chaîne de caractères représentant l'expression.

// POSTCONDITION : Retourne un entier résultat du calcul de l'expression

// EXCEPTION : Répercute une exception éventuellement levée par la Pile, l'Analyseur ou l'Automate

FONCTION CALCULER (eval : Evaluateur, expression : Chaine) : Entier

FAIRE

infixe, postfixe : File

rtn : Entier

//

infixe := ANALYSER (expression)

postfixe := CONVERTIR (infixe)

rtn := AUTOMATE (postfixe)

//

CALCULER := rtn

FIN\_FAIRE

**SOURCES :**

**2. Code Java :**

**package** tp.modele;

/\*\*

\* Représente une exception de type structure

\* **@author** Michel Devaux

\* **@version** 1.0.0

\*/

**public** **class** ExceptionStruct **extends** Exception{

/\*\*

\* Crée une instance de la classe avec un message générique.

\*/

**public** ExceptionStruct(){

**super**("Erreur non identifiée !");

}

/\*\*

\* Crée une instance de la classe avec un message personnalisé.

\* **@param** message Message décrivant l'exception levée.

\*/

**public** ExceptionStruct(String message){

**super**(message);

}

}

**package** tp.modele;

/\*\*

\* Représente une exception de type arithmétique ou de dépassement de capacité

\* **@author** Michel Devaux

\* **@version** 1.0.0

\*/

**public** **class** ExceptionMath **extends** Exception {

/\*\*

\* Crée une instance de la classe avec un message générique.

\*/

**public** ExceptionMath(){

**super**("Erreur non identifiée !");

}

/\*\*

\* Crée une instance de la classe avec un message personnalisé.

\* **@param** message Message décrivant l'exception levée.

\*/

**public** ExceptionMath(String message){

**super**(message);

}

}

**package** tp.modele;

/\*\*

\* Represente un element d'une expression

\* **@author** Michel Devaux

\* **@version** 1.0.0

\*/

**public** **class** TLexeme {

/\*\*

\* Définit la nature du lexeme

\*/

**public** **enum** Nature {

***operande***,

***operateur***,

***parenthese\_ouvrante***,

***parenthese\_fermante***;

}

/\*\*

\* Attributs

\*/

**public** String valeur;

**public** Nature nature;

/\*\*

\* Redéfinit la méthode toString()

\* **@return** Chaîne représentant la valeur de l'expression

\*/

**public** String toString() {

**return** **this**.valeur;

}

}

**package** tp.modele;

/\*\*

\* Représente une collection d'objets de type générique chaînés

\* sur le principe FIFO (premier entré, premier sorti).

\*

\* **@author** Michel DEVAUX

\* **@version** 1.1

\* **@param** <E> Type générique

\*/

**public** **class** File<E> {

/\*\*

\* Classe privée.</br> Représente un élément de la Pile composé <br/>

\* - d'une valeur de type générique <br/>

\* - d'une référence à l'élément suivant.

\*/

**private** **class** Element {

**public** E valeur;

**public** Element suivant;

}

/\*\*

\* Attributs

\*/

**private** Element tete;

**private** Element queue;

/\*\*

\* Constructeur <br/>

\* Crée un nouvel objet de type File.

\*/

**public** File() {

tete = **null**;

queue = **null**;

}

/\*\*

\* Teste si la File est vide.

\* **@return** TRUE si la file est vide.

\*/

**public** **boolean** fileVide() {

**return** (tete == **null**);

}

/\*\*

\* Renvoie l'élément le plus ancien de la File (le premier élément entré).

\* **@return** Objet de type générique.</br>

\* Retourne NULL si la file est vide.

\*/

**public** E lireFile() {

E elt = **null**;

**if** (!fileVide()) {

elt = tete.valeur;

}

**return** elt;

}

/\*\*

\* Enfile un nouvel élément à la queue de la File

\* **@param** elt Objet de type générique.

\*/

**public** **void** enfiler(E elt) {

Element element = **new** Element();

element.valeur = elt;

element.suivant = **null**;

**if** (fileVide()) {

tete = element;

queue = element;

} **else** {

queue.suivant = element;

queue = element;

}

}

/\*\*

\* Défile l'élément le plus ancien de la File.

\*/

**public** **void** defiler() {

**if** (!fileVide()) {

Element element = tete;

tete = tete.suivant;

element = **null**;

}

}

/\*\*

\* Redéfinit la méthode toString()

\* **@return** Chaîne représentant tous les éléments de la File.

\*/

**public** String toString() {

String str = "";

String separateur = " ";

//

Element element = tete;

**while** (element != **null**) {

str += element.valeur.toString() + separateur;

element = element.suivant;

}

str = str.substring(0, str.length() - separateur.length());

//

**return** str;

}

}

**package** tp.modele;

/\*\*

\* Représente une collection d'objets de type générique chaînés sur le principe

\* LIFO (dernier entré, premier sorti).

\*

\* **@author** Michel Devaux

\* **@version** 1.2

\* **@param** <E> Type générique

\*/

**public** **class** Pile<E> {

/\*\*

\* Classe privée.</br> Représente un élément de la Pile composé <br/>

\* - d'une valeur de type générique <br/>

\* - d'une référence à l'élément suivant.

\*/

**private** **class** Element {

**public** E valeur;

**public** Element suivant;

}

/\*\*

\* Attributs

\*/

**private** Element tete;

/\*\*

\* Constructeur <br/>

\* Crée un nouvel objet de type Pile.

\*/

**public** Pile() {

tete = **null**;

}

/\*\*

\* Teste si la pile est vide.

\* **@return** TRUE si la pile est vide.

\*/

**public** **boolean** pileVide() {

**return** (tete == **null**);

}

/\*\*

\* Renvoie l'élément supérieur de la pile (le dernier élément entré).

\* **@return** Objet de type générique.</br>

\* Retourne NULL si la pile est vide.

\*/

**public** E lirePile() {

E valeur = **null**;

**if** (!pileVide()) {

valeur = tete.valeur;

}

**return** valeur;

}

/\*\*

\* Empile un nouvel élément au sommet de la pile

\* **@param** elt Objet de type générique.

\*/

**public** **void** empiler(E valeur) {

Element element = **new** Element();

element.valeur = valeur;

element.suivant = tete;

tete = element;

}

/\*\*

\* Dépile le dernier élément entré dans la pile.

\*/

**public** **void** depiler() {

**if** (!pileVide()) {

Element element = tete;

tete = tete.suivant;

element = **null**;

}

}

/\*\*

\* Redéfinit la méthode toString()

\* **@return** Chaîne représentant les éléments entrés dans la Pile.

\*/

**public** String toString() {

String str = "";

String separateur = " ";

//

Element element = tete;

**while** (element != **null**) {

str += element.valeur.toString() + separateur;

element = element.suivant;

}

str = str.substring(0, str.length() - separateur.length());

//

**return** str;

}

}

**package** tp.controleur;

**import** tp.modele.\*;

**import** tp.modele.TLexeme.Nature;

/\*\*

\* Cette classe permet de convertir une expression infixée en une File d'objets

\* de type Lexeme.<br/> Cette classe n'est pas publique.

\*

\* **@author** Michel Devaux

\* **@version** 1.1

\*/

**class** Analyseur {

// Constantes

**private** **final** String CHIFFRES = "0123456789";

**private** **final** String SIGNES = "+-\*/";

**private** **final** String PARENTHESES = "()";

// File contenant en fin de traitement les éléments de l'expression.

**private** File<TLexeme> infixe;

// Structure schématique de l'expression. Ex : ((6\*5)+3) -> ((n.n).n)

**private** String structure;

// Autorise ou non quelques tolérances dans la gestion des parenthèses

**private** **boolean** tolerant = **true**;

/\*\*

\* Getter de l'attribut "tolerant".

\* **@return** TRUE si "tolerant" est actif.

\*/

**public** **boolean** isTolerant() {

**return** tolerant;

}

/\*\*

\* Active ou non la tolérance de l'Analyseur

\* **@param** tolerant

\*/

**public** **void** setTolerant(**boolean** tolerant) {

**this**.tolerant = tolerant;

}

/\*\*

\* Obtient, en les analysant, la liste des elements qui composent

\* l'expression et les range dans une File d'éléments de type TLexeme.

\*

\* **@return** Instance de la classe File.

\* **@param** expression Expression de type String à traiter.

\*

\* **@throws** Exception

\* Retourne une exception de structure, arithmétique ou de dépassement de

\* capacité

\*/

**public** File<TLexeme> getFileInfixe(String expression)

**throws** ExceptionStruct, ExceptionMath {

// Déclarations

infixe = **new** File<TLexeme>();

TLexeme elt;

structure = "";

**int** i = 0;

//

// Expression vide

**if** (expression.equals("")) {

**throw** **new** ExceptionStruct("L'expression est vide.");

}

//

**while** (i < expression.length()) {

// Caractère en cours

String str = expression.substring(i, i + 1);

//

// Traitement des chiffres

**if** (CHIFFRES.indexOf(str) != -1) {

// Former le nombre en recherchant les chiffres suivants

**while** (i + 1 < expression.length()

&& CHIFFRES.indexOf(expression

.substring(i + 1, i + 2)) != -1) {

i++;

str += expression.substring(i, i + 1);

}

// Tester si le nombre ne dépasse pas la capacité d'un

// entier

**try** {

**int** test = Integer.*parseInt*(str);

} **catch** (NumberFormatException e) {

// Lever une ExceptionMath

**throw** **new** ExceptionMath("L'expression contient

des opérandes trop grands.");

}

elt = **new** TLexeme();

elt.valeur = str;

elt.nature = Nature.***operande***;

infixe.enfiler(elt);

// Ajouter la représentation d'un opérande à la

// structure

structure += "n";

}

//

// Traitement des opérateurs

**else** **if** (SIGNES.indexOf(str) != -1) {

// Ajouter l'opérateur dans la File

elt = **new** TLexeme();

elt.valeur = str;

elt.nature = Nature.***operateur***;

infixe.enfiler(elt);

// Ajouter la représentation d'un opérateur à la

// structure

structure += ".";

}

//

// Traitement des parenthèses

**else** **if** (PARENTHESES.indexOf(str) != -1) {

// Ajouter la parenthèse dans la File

elt = **new** TLexeme();

elt.valeur = str;

elt.nature = (str.equals("("))

? Nature.***parenthese\_ouvrante***

: Nature.***parenthese\_fermante***;

infixe.enfiler(elt);

// Ajouter la parenthèse à la structure

structure += str;

}

//

// Ignorer les espaces

**else** **if** (str.equals(" ")) {

// On ne fait rien là...

}

//

// Sinon str = caractère non autorisé

**else** {

**throw** **new** ExceptionStruct("L'expression contient des

caractères non autorisés.");

}

// Incrémenter le pointeur

i++;

}

// Vérifier la struture

**if** (!verifierStructure()) {

**throw** **new** ExceptionStruct("L'expression n'est pas correcte."

+ "\nVérifiez le compte des parenthèses, opérandes

et opérateurs.");

}

// Retourner la File

**return** infixe;

}

/\*\*

\* Vérifie la structure de l'expression en la réduisant

\* progressivement.<br/>

\* La structure complètement réduite doit être égale à "n".<br/>

\* Exemple : (n.(n.n)) -> (n.n) -> n

\*

\* **@return** TRUE si la structure est correcte.

\*/

**private** **boolean** verifierStructure() {

// Déclarations

String chaine = structure;

String recherche = "(n.n)";

String remplace = "n";

//

// Tolérer optionnellement en entrée quelques libertés avec la

// syntaxe

**if** (tolerant) {

// Ajoute des parenthèses autour de l'expression : 5\*2 -> (5\*2)

chaine = "(" + chaine + ")";

// Supprime les parenthèses excédentaires autour d'un nombre

// seul

chaine = chaineCorrigee(chaine);

}

//

// Boucle principale assurant les conversions : (n.n) -> n

**while** (chaine.indexOf(recherche) != -1) {

// Remplacer

chaine = chaine.replace(recherche, remplace);

// Tolérance

**if** (tolerant) chaine = chaineCorrigee(chaine);

}

// Tolérer optionnellement en sortie quelques libertés avec la

// syntaxe

**if** (tolerant) chaine = chaineCorrigee(chaine);

//

// Affiche le résultat dans la console. Doit être égal à : n

**boolean** resultat = chaine.equals(remplace);

String format = "RESULTAT DE L'ANALYSE LEXICALE : %s = %s";

System.***out***.println(String.*format*(format, structure, chaine));

//

// Renvoie le résultat de l'analyse

**return** resultat;

}

/\*\*

\* Supprime les parenthèses excédentaires autour d'un nombre seul <br/>

\* (n) -> n

\* **@param** chaine

\* **@return**

\*/

**private** String chaineCorrigee(String chaine) {

**while** (chaine.indexOf("(n)") != -1) {

chaine = chaine.replace("(n)", "n");

}

**return** chaine;

}

}

**package** tp.controleur;

**import** tp.modele.\*;

**import** tp.modele.TLexeme.Nature;

/\*\*

\* Cette classe contient des méthodes de traitement et de resolution d'une

\* expression infixee.

\*

\* **@author** Michel Devaux

\* **@version** 1.4

\*/

**public** **class** Evaluateur **implements** IEvaluateur {

// Instance de l'interface de la classe (Singleton)

**private** **static** IEvaluateur *evaluateur*;

// Instance de l'Analyseur

**private** Analyseur analyseur;

/\*\*

\* Le constructeur privé protège la classe qui n'est accessible que par

\* son Singleton.

\* Ce Singleton renvoie l'Instance unique de l'Interface liée.

\*/

**private** Evaluateur() {

analyseur = **new** Analyseur();

}

/\*\*

\* Singleton de l'Evaluateur

\* **@return** Instance unique de type IEvaluateur.

\*/

**public** **static** IEvaluateur getEvaluateur() {

**if** (*evaluateur* == **null**) {

*evaluateur* = **new** Evaluateur();

}

**return** *evaluateur*;

}

/\*\*

\* Active ou non la tolérance de l'Analyseur

\* **@param** tolerant

\*/

**public** **void** setTolerance(**boolean** tolerant) {

analyseur.setTolerant(tolerant);

}

/\*\*

\* Utilise une instance de la classe Analyseur pour analyser une expression

\* et placer les éléments qui la composent dans une File d'éléments de type

\* Lexeme.

\*

\* **@param** expression Chaine représentant l'expression à analyser

\* **@return** File d'éléments de type TLexeme

\* **@throws** Exception Répercute les exceptions levées par l'Analyseur.

\* **@see** Analyseur#getFileInfixe(String)

\*/

**private** File<TLexeme> analyser(String expression)

**throws** ExceptionStruct, ExceptionMath {

File<TLexeme> infixe = analyseur.getFileInfixe(expression);

**return** infixe;

}

/\*\*

\* Convertit une expression infixée en expression post-fixée.<br/>

\* Les paramètres d'entrée et sortie sont des Files.

\*

\* **@param** file File d'éléments de type TLexeme

\* **@return** File d'éléments de type TLexeme

\*/

**private** File<TLexeme> convertir(File<TLexeme> infixe) {

// Création de la File de sortie

File<TLexeme> postfixe = **new** File<TLexeme>();

// Création de la Pile qui stockera parenthèses et opérateurs

Pile<TLexeme> maPile = **new** Pile<TLexeme>();

// Déclarations

TLexeme elt;

//

// C'est parti !!!

**while** (!infixe.fileVide()) {

// Lire l'élément de tête et l'enlever de la File

elt = infixe.lireFile();

infixe.defiler();

// En fonction de la nature de l'élément...

**switch** (elt.nature) {

**case** ***operande***:

// C'est un opérande...

// On l'ajoute à la File postfixée.

postfixe.enfiler(elt);

**break**;

**case** ***operateur***:

// C'est un opérateur...

// On l'empile.

maPile.empiler(elt);

**break**;

**case** ***parenthese\_ouvrante***:

// C'est une parenthèse ouvrante...

// On l'empile.

maPile.empiler(elt);

**break**;

**case** ***parenthese\_fermante***:

// C'est une parenthèse fermante...

// Dépiler la pile jusque la parenthèse ouvrante.

// Les éléments lus sont ajoutés à la File de sortie.

elt = maPile.lirePile();

**while** (elt.nature != Nature.***parenthese\_ouvrante***) {

postfixe.enfiler(elt);

maPile.depiler();

elt = maPile.lirePile();

}

// Dépiler la parenthèse ouvrante

maPile.depiler();

**break**;

}

}

// Vider le reste de la pile (les opérateurs restants).

**while** (!maPile.pileVide()) {

elt = maPile.lirePile();

postfixe.enfiler(elt);

maPile.depiler();

}

// Retourner la File

**return** postfixe;

}

/\*\*

\* L'automate retourne le calcul d'une expression post-fixée passée sous la

\* forme d'une File d'éléments de type TLexeme.

\*

\* **@param** postfixe File d'éléments de type TLexeme ordonnés en

\* expression post-fixée.

\* **@return** Entier résultant du calcul de l'expression passée en paramètre.

\* **@throws** Exception Répercute l'exception levée par une division par zéro

\* ou par un dépassement de capacité.

\*/

**private** **int** automate(File<TLexeme> postfixe) **throws** ExceptionMath {

// Création de la Pile qui stockera les opérandes

Pile<TLexeme> maPile = **new** Pile<TLexeme>();

// Déclarations

**int** val1;

**int** val2;

**int** val3;

**long** lVal;

TLexeme elt;

//

// C'est parti !!!

**while** (!postfixe.fileVide()) {

// Lire l'élément de tête et l'enlever de la File.

elt = postfixe.lireFile();

postfixe.defiler();

// En fonction de la nature de l'élément...

**switch** (elt.nature) {

**case** ***operande***:

// C'est un opérande...

// On l'empile.

maPile.empiler(elt);

**break**;

**case** ***operateur***:

// C'est un opérateur...

// Dépiler les 2 opérandes concernés par l'opération.

val2 = Integer.*parseInt*(maPile.lirePile().valeur);

maPile.depiler();

val1 = Integer.*parseInt*(maPile.lirePile().valeur);

maPile.depiler();

// Effectuer le calcul approprié.

**if** (elt.valeur.equals("+")) {

// Addition

val3 = val1 + val2;

// Vérifier si le résultat ne provoque pas un

// débordement de int

**if** (((**long**)val1 + (**long**)val2) != val3){

**throw** **new** ExceptionMath("L'expression

provoque un dépassement de capacité");

}

}

**else** **if** (elt.valeur.equals("-")) {

// Soustraction

val3 = val1 - val2;

// Vérifier si le résultat ne provoque pas un

// débordement de int

**if** (((**long**)val1 - (**long**)val2) != val3){

**throw** **new** ExceptionMath("L'expression

provoque un dépassement de capacité");

}

}

**else** **if** (elt.valeur.equals("\*")) {

// Multiplication

val3 = val1 \* val2;

// Vérifier si le résultat ne provoque pas un

// débordement de int

**if** (((**long**)val1 \* (**long**)val2) != val3){

**throw** **new** ExceptionMath("L'expression

provoque un dépassement de capacité");

}

}

**else** {

// Division

**if** (val2 != 0) {

val3 = val1 / val2;

}

**else** {

// Division par zéro...

// Renvoyer une ExceptionMath

**throw** **new** ExceptionMath("L'expression

provoque une division par zéro.");

}

// Vérifier si le résultat ne provoque pas un

// débordement de int

**if** (((**long**)val1 / (**long**)val2) != val3){

**throw** **new** ExceptionMath("L'expression

provoque un dépassement de capacité");

}

}

// Affecter à elt une nouvelle instance de TLexeme

// ayant pour valeur le résultat de l'opération.

elt = **new** TLexeme();

elt.valeur = "" + val3;

elt.nature = Nature.***operande***;

// Empiler le résulat elt.

maPile.empiler(elt);;

**break**;

**default**:

**break**;

}

}

// Récupérer le résultat final de l'opération dans la Pile

elt = maPile.lirePile();

val1 = Integer.*parseInt*(elt.valeur);

// Retourner la valeur en tant qu'entier

**return** val1;

}

/\*\*

\* Effectue le calcul d'une expression passée en paramètre

\* en utilisant successivement l'analyseur, le convertisseur

\* et l'automate à Pile.

\*

\* **@param** expression Expression infixée. Exemple : ((5+3)\*6)

\* **@return** Résultat de type entier.

\* **@throws** Exception Répercute l'exception levée par l'Analyseur ou

\* par une des trois méthodes invoquées.

\*/

@Override

**public** **int** traitement(String expression)

**throws** ExceptionStruct, ExceptionMath {

**int** rtn = 0;

File<TLexeme> maFile = **null**;

System.***out***.println(expression);

// Analyse de l'expression.

maFile = analyser(expression);

System.***out***.println("FILE EVALUATEUR : " + maFile);

// Conversion en expression postfixée.

maFile = convertir(maFile);

System.***out***.println("FILE CONVERTISSEUR : " + maFile);

// Utilisation de l'automate pour effectuer le calcul.

rtn = automate(maFile);

System.***out***.println("RESULTAT AUTOMATE : " + rtn);

// Retourner le résulat.

**return** rtn;

}

}

**package** tp.controleur;

**import** tp.modele.\*;

/\*\*

\* Interface de l'Evaluateur.<br/>

\* Ne laisse apparaître que la seule méthode Calculer.

\*

\* **@author** Michel

\* **@version** 1.0

\*/

**public** **interface** IEvaluateur {

**public** **int** traitement(String expression)

**throws** ExceptionStruct, ExceptionMath;

**public** **void** setTolerance(**boolean** tolerance);

}

**package** tp.controleur;

**import** tp.vue.\*;

/\*\*

\* Classe de lancement du programme.</br> Contient la méthode "main".

\*

\* **@author** Michel Devaux

\* **@version** 1.0

\*/

**public** **class** MainTP {

**private** **static** **boolean** *modeGraphique* = **true**;

/\*\*

\* Première méthode lancée par le programme.

\*

\* **@param** args optionnellement passés à la méthode.

\*/

**public** **static** **void** main(String[] args) {

**if** (*modeGraphique*) {

// Mode Graphique

**new** IHMGraphique();

} **else** {

// Mode Console

IHMConsole ihm = **new** IHMConsole();

ihm.saisir();

}

}

}

**package** tp.vue;

**import** tp.controleur.\*;

**import** tp.modele.\*;

**import** java.util.Scanner;

/\*\*

\* Première interface de saisie en mode console.

\*

\* **@author** Michel Devaux

\* **@version** 1.0.0

\*/

**public** **class** IHMConsole {

**public** **void** saisir() {

Scanner sc = **new** Scanner(System.***in***);

String expression = sc.next();

//

**try** {

IEvaluateur evaluateur = Evaluateur.*getEvaluateur*();

**int** resultat = evaluateur.traitement(expression);

System.***out***.println(expression + " = " + resultat);

} **catch** (ExceptionStruct e) {

System.***out***.println(e.getMessage());

} **catch** (ExceptionMath e) {

System.***out***.println(e.getMessage());

}

sc.close();

}

}

**package** tp.vue;

**import** java.awt.\*;

**import** java.awt.event.\*;

**import** javax.swing.\*;

**import** tp.controleur.\*;

/\*\*

\* Fenêtre de saisie utilisant Swing.

\*

\* **@author** Michel Devaux

\* **@version** 1.6

\*/

**public** **class** IHMGraphique **extends** JFrame {

**private** Font font = **new** Font("Calibri", Font.***PLAIN***, 16);

// Principaux contrôles

**private** TextFieldFiltre txtExpression;

**private** JCheckBox chkFiltre;

**private** TextAreaScrollable zoneResultats;

// ActionListener unique pour tous les contrôles

**private** ActionEvnt actionEvnt = **new** ActionEvnt();

// Identificateurs : pour chacun des contrôles invoquant l'ActionListener

**private** **final** String TXTEXPRESSION\_EVNT = "0001";

**private** **final** String BTNVALIDER\_EVNT = "0002";

**private** **final** String BTNEFFACER\_EVNT = "0003";

**private** **final** String CHKFILTRE\_EVNT = "0004";

/\*\*

\* Création de la fenêtre et de ses composants

\*/

**public** IHMGraphique() {

// Quitter l'application à la fermeture de la fenêtre

setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

// Taille

setPreferredSize(**new** Dimension(600, 400));

// Nom

setTitle("TP-NFA032-Michel Devaux");

// Pas de redimensionnement

setResizable(**false**);

// Définir un layout au centre avec 20 px de margin H et V

setLayout(**new** FlowLayout(FlowLayout.***CENTER***, 20, 20));

//

// Définir le panel principal

JPanel panel = **new** JPanel(**new** FlowLayout(FlowLayout.***CENTER***,0,0));

// Definir la taille préférée du panel principal

panel.setPreferredSize(**new** Dimension(540, 320));

add(panel);

//

// Les contrôles...

// Titre Expression

JLabel lbl = **new** JLabel("Entrez ci-dessous votre expression, puis

validez :");

lbl.setPreferredSize(**new** Dimension(500, 40));

lbl.setFont(font);

panel.add(lbl);

// Champ d'édition Expression

txtExpression = **new** TextFieldFiltre();

txtExpression.setPreferredSize(**new** Dimension(500, 40));

txtExpression.setHorizontalAlignment(JTextField.***CENTER***);

txtExpression.setFont(font);

txtExpression.setActionCommand(TXTEXPRESSION\_EVNT);

txtExpression.addActionListener(actionEvnt);

txtExpression.setFiltrer(**false**);

txtExpression.setCaracteresAutorises("0123456789+-\*/()");

panel.add(txtExpression);

// Label vide (pour aérer)

lbl = **new** JLabel(" ");

lbl.setPreferredSize(**new** Dimension(500, 10));

panel.add(lbl);

// Case à cocher Filtrer

chkFiltre = **new** JCheckBox();

chkFiltre.setText("Filtrer les caractères de l'expression en

entrée");

chkFiltre.setFont(font);

chkFiltre.setPreferredSize(**new** Dimension(340, 30));

chkFiltre.setSelected(txtExpression.isFiltrer());

chkFiltre.setActionCommand(CHKFILTRE\_EVNT);

chkFiltre.addActionListener(actionEvnt);

panel.add(chkFiltre);

// Bouton Effacer

JButton btn = **new** JButton();

btn.setText ("Effacer");

btn.setPreferredSize(**new** Dimension(75, 30));

btn.setActionCommand(BTNEFFACER\_EVNT);

btn.addActionListener(actionEvnt);

panel.add(btn);

// Label vide (pour aérer)

lbl = **new** JLabel("");

lbl.setPreferredSize(**new** Dimension(10, 20));

panel.add(lbl);

// Bouton Valider

btn = **new** JButton();

btn.setText ("Valider");

btn.setPreferredSize(**new** Dimension(75, 30));

btn.setActionCommand(BTNVALIDER\_EVNT);

btn.addActionListener(actionEvnt);

panel.add(btn);

// Label vide (pour aérer)

lbl = **new** JLabel("");

lbl.setPreferredSize(**new** Dimension(500, 20));

panel.add(lbl);

// Titre Resultats

lbl = **new** JLabel("Résultats :");

lbl.setFont(font);

lbl.setPreferredSize(**new** Dimension(500, 30));

panel.add(lbl);

// TextAreaScrollable Resultats

zoneResultats = **new** TextAreaScrollable(**new** Dimension(500, 135));

JTextArea area = zoneResultats.getTextArea();

area.setFont(font);

area.setEditable(**false**);

panel.add(zoneResultats);

// Laisser à Swing le soin d'optimiser la fenêtre

pack();

// Centrer la fenêtre dans l'écran

setLocationRelativeTo(**null**);

// Afficher la fenêtre

setVisible(**true**);

}

/\*\*

\* Lance le calcul de l'expression.<br/>

\* Le résultat est placé dans zoneResultats.

\* La méthode gère les erreurs renvoyées par l'Evaluateur (fenêtre d'erreur)

\*/

**private** **void** valider() {

// Récupère l'expression dans le JTextField

String expression = txtExpression.getText();

// Try catch pour gérer à cet endroit une erreur éventuelle

**try** {

// Récupère le singleton de la classe Evaluateur

IEvaluateur evaluateur = Evaluateur.*getEvaluateur*();

// Mettre la tolérance à true ou à false (option)

// evaluateur.setTolerance(false);

**int** calcul = evaluateur.traitement(expression);

// Récupérer le JTextArea de la zoneResultats

JTextArea area = zoneResultats.getTextArea();

// Ajouter le résultat

String resultat = expression + " = " + calcul;

area.setText(area.getText() + resultat + "\n");

} **catch** (Exception e) {

// Une erreur ! Afficher le message

// dans une fenêtre de dialogue

JOptionPane.*showMessageDialog*(**this**,

e.getMessage(), "Calcul non réalisable", JOptionPane.***ERROR\_MESSAGE***);

}

}

/\*\*

\* Effacer le contenu du champ d'édition

\*/

**private** **void** effacer() {

txtExpression.setText("");

}

/\*\*

\* Classe privée.<br/>

\* Représente un évènement de type ActionListener

\*

\* **@author** Michel

\* **@version** 1.1

\*/

**private** **class** ActionEvnt **implements** ActionListener {

@Override

**public** **void** actionPerformed(ActionEvent e) {

// Récupérer l'ActionCommand pour identifier la source

String action = e.getActionCommand();

// Si clic sur Valider ou touche ENTER dans txtExpression

**if** (action.equals(BTNVALIDER\_EVNT) ||

action.equals(TXTEXPRESSION\_EVNT) ) {

valider();

}

// Si clic sur chkFiltre

**if** (action.equals(CHKFILTRE\_EVNT)) {

txtExpression.setFiltrer(chkFiltre.isSelected());

}

// Si clic sur Effacer

**if** (action.equals(BTNEFFACER\_EVNT)) {

effacer();

}

}

}

}

**package** tp.vue;

**import** java.awt.event.KeyEvent;

**import** java.awt.event.KeyListener;

**import** javax.swing.JTextField;

/\*\*

\* Représente un JTextField avec filtrage des caractères

\*

\* **@author** Michel Devaux

\* **@version** 1.1

\*/

**public** **class** TextFieldFiltre **extends** JTextField {

**private** String caracteresAutorises = "";

**private** **boolean** filtrer = **false**;

**public** TextFieldFiltre() {

// Supprimer le Copier/Coller

// setTransferHandler(null);

// Ajouter un KeyListener pour filtrer les caractères entrants

addKeyListener(**new** KeyListener() {

@Override

**public** **void** keyTyped(KeyEvent e) {

**if** (filtrer && !caracteresAutorises.equals("")) {

**char** c = e.getKeyChar();

**if** (caracteresAutorises.indexOf("" + c) == -1)

e.consume();

}

}

@Override

**public** **void** keyReleased(KeyEvent e) {}

@Override

**public** **void** keyPressed(KeyEvent e) {}

});

}

/\*\*

\* Renvoie les caractères autorisés par le filtre

\* **@return** Chaîne contenant les caractères autorisés

\*/

**public** String getCaracteresAutorises() {

**return** caracteresAutorises;

}

/\*\*

\* Détermine les caractères autorisés par le filtre

\* **@param** caracteresAutorises Chaîne contenant les caractères autorisés

\*/

**public** **void** setCaracteresAutorises(String caracteresAutorises) {

**this**.caracteresAutorises = caracteresAutorises;

}

/\*\*

\* Renvoie l'état du filtrage

\* **@return** TRUE si le filtre est actif

\*/

**public** **boolean** isFiltrer() {

**return** filtrer;

}

/\*\*

\* Active ou désactive le filtre

\* **@param** filtrer Vrai : active le filtre

\*/

**public** **void** setFiltrer(**boolean** filtrer) {

**this**.filtrer = filtrer;

**if** (filtrer)

filtrerTexte();

}

/\*\*

\* Filtre le texte déjà entré pour ne conserver que les caractères autorisés.

\*/

**private** **void** filtrerTexte() {

String texte = getText();

String nouveau = "";

**for** (**char** c : texte.toCharArray()) {

**if** (caracteresAutorises.indexOf("" + c) != -1) {

nouveau += c;

}

}

setText(nouveau);

}

}

**package** tp.vue;

**import** java.awt.\*;

**import** javax.swing.\*;

/\*\*

\* Représente un Panel contenant un TextArea associé à un ScrollPane.

\*

\* **@author** Michel Devaux

\* **@version** 1.0

\*/

**public** **class** TextAreaScrollable **extends** JPanel {

**private** JTextArea textArea;

/\*\*

\* Initialise une instance de la classe.

\* **@param** dimensions Taille de la zone.

\*/

**public** TextAreaScrollable(Dimension dimensions) {

setPreferredSize(dimensions);

// Créer un layout dans lequel le TextArea prendra tout l'espace

GridLayout layout = **new** GridLayout(1,1);

setLayout(layout);

// Créer le JTextArea

textArea = **new** JTextArea();

// Créer le ScrollPane associé

JScrollPane scroll = **new** JScrollPane(textArea);

scroll.setHorizontalScrollBarPolicy(JScrollPane

.***HORIZONTAL\_SCROLLBAR\_AS\_NEEDED***);

scroll.setVerticalScrollBarPolicy(JScrollPane

.***VERTICAL\_SCROLLBAR\_AS\_NEEDED***);

// Ajouter le ScrollPane au composant

add(scroll);

}

/\*\*

\* Récupère le TextArea associé au composant.<br/>

\* Permet de paramétrer le TextArea : fonte, couleur, éditable, etc.

\* **@return**

\*/

**public** JTextArea getTextArea() {

**return** textArea;

}

}

**package** tp.controleur;

**import** org.junit.Test;

**import** org.junit.Assert;

**import** tp.modele.ExceptionMath;

**import** tp.modele.ExceptionStruct;

/\*\*

\* Batterie de tests JUnit

\*

\* **@author** Michel Devaux Version 1.0

\*/

**public** **class** EvaluateurTest {

@Test(timeout = 1000)

**public** **void** testCalculer01()

**throws** ExceptionStruct, ExceptionMath {

// assetEquals(valeur attendue, valeur calculée, tolérance)

Assert.*assertEquals*(5, Evaluateur.getEvaluateur()

.traitement("5"), 0);

}

@Test(timeout = 1000)

**public** **void** testCalculer02()

**throws** ExceptionStruct, ExceptionMath {

Assert.*assertEquals*(5558, Evaluateur.getEvaluateur()

.traitement("(5555+3)"), 0);

}

@Test(timeout = 1000)

**public** **void** testCalculer03()

**throws** ExceptionStruct, ExceptionMath {

Assert.*assertEquals*(8, Evaluateur.getEvaluateur()

.traitement("(8)"), 0);

}

@Test(timeout = 1000)

**public** **void** testCalculer04()

**throws** ExceptionStruct, ExceptionMath {

Assert.*assertEquals*(5558,Evaluateur.getEvaluateur()

.traitement("(3+5555)"), 0);

}

@Test(timeout = 1000)

**public** **void** testCalculer05()

**throws** ExceptionStruct, ExceptionMath {

Assert.*assertEquals*(25, Evaluateur.getEvaluateur()

.traitement("(5\*(2+3))"), 0);

}

@Test(timeout = 1000)

**public** **void** testCalculer06()

**throws** ExceptionStruct, ExceptionMath {

Assert.*assertEquals*(65536, Evaluateur.getEvaluateur()

.traitement("((125+3)\*512)"), 0);

}

@Test(timeout = 1000)

**public** **void** testCalculer07()

**throws** ExceptionStruct, ExceptionMath {

Assert.*assertEquals*(-2, Evaluateur.getEvaluateur()

.traitement("(2\*((((53-52)\*1)/5)+(6-7)))"), 0);

}

@Test(expected = ExceptionMath.**class**, timeout = 1000)

**public** **void** testCalculer08()

**throws** ExceptionStruct, ExceptionMath {

**int** res = Evaluateur.getEvaluateur()

.traitement("(2\*((((3-2)\*1)/(17-((2\*3)+11)))+(6-7)))");

}

@Test(timeout = 1000)

**public** **void** testCalculer09()

**throws** ExceptionStruct, ExceptionMath {

Assert.*assertEquals*(-2,Evaluateur.getEvaluateur()

.traitement("(2\*(( ((3-2)\*1)/5)+(6-7)))"), 0);

}

@Test(expected = ExceptionStruct.**class**, timeout = 1000)

**public** **void** testCalculer10()

**throws** ExceptionStruct, ExceptionMath {

**int** res = Evaluateur.getEvaluateur()

.traitement("(2\*(((((3-2)\*1)/5)+(6-7)))");

}

@Test(expected = ExceptionStruct.**class**, timeout = 1000)

**public** **void** testCalculer11()

**throws** ExceptionStruct, ExceptionMath {

**int** res = Evaluateur.getEvaluateur()

.traitement("(2\*((((3-2)\*1)/5))+(6-7))))");

}

// TESTS COMPLEMENTAIRES (PERSONNELS)

@Test(expected = ExceptionStruct.**class**, timeout = 1000)

**public** **void** testCalculer12()

**throws** ExceptionStruct, ExceptionMath {

**int** res = Evaluateur.getEvaluateur()

.traitement("()");

}

@Test(expected = ExceptionStruct.**class**, timeout = 1000)

**public** **void** testCalculer13()

**throws** ExceptionStruct, ExceptionMath {

**int** res = Evaluateur.getEvaluateur()

.traitement("(5++3");

}

@Test(expected = ExceptionStruct.**class**, timeout = 1000)

**public** **void** testCalculer14()

**throws** ExceptionStruct, ExceptionMath {

**int** res = Evaluateur.getEvaluateur()

.traitement("(-53)");

}

@Test(timeout = 1000)

**public** **void** testCalculer15()

**throws** ExceptionStruct, ExceptionMath {

Assert.*assertEquals*(523, Evaluateur.getEvaluateur()

.traitement("((523))"), 0);

}

@Test(expected = ExceptionMath.**class**, timeout = 1000)

**public** **void** testCalculer16()

**throws** ExceptionStruct, ExceptionMath {

**int** res = Evaluateur.getEvaluateur()

.traitement("(99999999999 + 2)");

}

@Test(expected = ExceptionMath.**class**, timeout = 1000)

**public** **void** testCalculer17()

**throws** ExceptionStruct, ExceptionMath {

**int** res = Evaluateur.getEvaluateur()

.traitement("((123456\*123456)\*123456)");

}

}