RÉALISATION D'UN CALCULATEUR EN NOTATION POST-FIXEE:

1. Présentation :

Quand on écrit une expression arithmétique, on utilise généralement une notation dite infixée : les opérateurs +, /, - et * sont placés entre les deux expressions auxquelles ils s'appliquent. Autrement dit, les opérateurs sont entourés par les opérandes. On écrit par exemple : (3+4) * (7-5). Cette notation est la plus lisible pour un humain, mais elle n'est pas la plus simple pour un ordinateur. Il est nécessaire d'utiliser des parenthèses et la technique d'évaluation doit donc en tenir compte. Cependant, il existe donc d'autres notations qui rendent les calculs plus simples. La notation post-fixée, aussi appelée notation polonaise inverse, permet de placer les opérateurs après les opérandes : 3 4 + 7 5 - * .
Le *Tableau 1* présente des exemples d'expressions en notation in-fixée et leurs équivalences en notation post-fixée.

Tableau 1 : Expressions en notation in-fixée et post-fixée

Expression infixée	Expression post-fixée
a + b	a b +
a + b * c	a b c * +
a * b + c	a b * c +
(a + b) * c	a b + c *
(a - b) * (c + d)	a b - c d + *
(a + b) * (c – d/e) + f	a b + c d e / - * f +

Le but de ce projet est de réaliser en langage Java, un programme calculant la valeur d'une expression arithmétique notée en post-fixée.

2. Spécification du projet :

a- Syntaxe des expressions :

Les expressions seront formées des items suivants :

- ✔ des nombres décimaux positifs : 123.45
- ✔ des 5 opérateurs binaires : + , –, *, / , et puissance (^);
- ✔ deux opérateurs unaires : neg et cos;
- ✔ des identificateurs que l'on supposera composés d'un seul caractère;

b- Programme principal :

L'interface du programme proposera successivement :

1- Le choix de l'affichage des calculs.

Et indéfiniment :

2- La saisie d'une expression arithmétique, par exemple :

Expression à calculer : 2 4 3 2 ^ * +

**Expression à calculer : a b + d c + a / *

3- La saisie des valeurs des identificateurs présents dans l'expression saisie, a, b, d et c, dans le cas de la deuxième expression.
Dans le cas d'une expression avec opérateurs (ou fonctions), qui est censée prendre n arguments en paramètres, il faudra vérifier l'existence de ces n arguments. Nous supposerons également que la première ligne représente l'expression à calculer et les n lignes suivantes symbolisent les n variables de l'expression en question, dans leur ordre d'apparition. Par exemple pour calculer 45/9, deux identificateurs sont utilisés :

Expression à calculer : b a /
Expression de b : 45
Expression de a : 9

Il est à noter que la saisie d'une nouvelle expression initialisera toutes les variables employées dans le calcul de l'expression précédente.

4- A la suite d'une saisie correcte, le programme affichera la valeur de l'expression calculée en fonction des valeurs des identificateurs.

5- Pour enregistrer le résultat de calcul, un fichier nommé « **Resultats.txt** » devra être créé en mode ajout. Chaque équation saisie sera écrite dans ce fichier en notation infixée avec son résultat de calcul.

6- Dans le cas où une erreur de saisie est détectée lors de la lecture de l'expression, le programme invitera l'utilisateur à la ressaisir. De même lors de la saisie d'une valeur erronée pour un identificateur, le programme demandera à nouveau la saisie de l'expression (Retour au point 1).

Les erreurs des tests non valides seront également stockées dans un fichier nommé « **Erreurs.txt** ». Quelques exemples d'erreurs sont présentés ci-dessous:

Expression : 1 +
Erreur: Argument manquant.

Expression : 1 < 2
Erreur: Caractère invalide (<).

Expression : 2 0 /
Erreur: Division par zéro.

L'**Annexe A** présente un exemple d'utilisation du logiciel.

Note : La classe Scanner sera utilisée pour lire une expression saisie depuis l'entrée standard (System.in). Les opérandes saisies sont des nombres réels (float ou double). Si l'utilisateur saisit la valeur 1.5, une exception de type **java.util.InputMismatchException** est levée, car un nombre réel en France s'écrit à l'aide d'une virgule, alors qu'aux États-Unis, on utilise le point. Ainsi, l'appel de useLocale est nécessaire pour que Scanner lise les flottants avec un point décimal et non une virgule, si jamais la machine est installée en français (useLocale(Locale.ENGLISH)).

c- Algorithme de calcul :

L'algorithme général de calcul d'une expression en notation post-fixée s'effectue simplement à l'aide d'une pile.
Une pile est une structure de données dans laquelle on peut ajouter et supprimer des éléments suivant la règle du dernier entré premier sorti (LIFO de l'anglais : Last In First Out). Le nom de la pile vient d'une analogie avec une pile d'assiettes (où l'on poserait et l'on prendrait toujours des assiettes sur le dessus de la pile). Les opérations de base, appelées primitives de gestion des piles sont :

- **initialiser :** pour créer une pile vide,
- **estVide :** renvoie vrai si la pile est vide, faux sinon,
- **empiler :** cette fonction permet d'ajouter un élément au sommet de la pile,
- **depiler :** cette fonction supprime le sommet de la pile. L'élément supprimé est retourné par la fonction **depiler** pour pouvoir l'utiliser.

Par la suite, on supposera que les termes de l'expression sont entrecoupés de séparateurs ; ici le seul séparateur sera le caractère espace et les termes de l'expression seront soit un nombre flottant, soit un identificateur, soit un opérateur.
L'algorithme de calcul s'explicite comme suit :

- Si le terme courant est un nombre, le mettre sur la pile.
- Si le terme courant est un opérateur, récupérer les opérandes, effectuer l'opération, puis mettre le résultat sur la pile.
- L'algorithme s'arrête lorsqu'il n'y a plus de terme à extraire d'une ligne et que la pile ne contient plus qu'un seul élément : le résultat.

Prenons par exemple l'expression 2 3 ^ 5 4 * - (correspondant à l'expression infixée suivante : ((2^3) - (5 * 4)). On effectue les traitements suivants, écrits en pseudo-code :

Tableau 2 : Pseudo-code du calcul de l'expression 2 3 ^ 5 4 * -

Traitements	État de la pile
Terme : 2 **empiler(2)**	**2**
Terme : 3 **empiler(3)**	**3 2**
Terme : ^ **depiler(), depiler(),** **calculer(2^3),** **empiler(8)**	**8**
Terme : 5 **empiler(5)**	**5 8**

Terme :: 4 **empiler(4)**	**4 5 8**
Terme : * **depiler(), depiler(),** **calculer(4*5),** **empiler(20)**	**20 8**
Terme : - **depiler(), depiler(),** **calculer(20-8),** **empiler(-12)**	**-12**

S'il s'agit d'une variable (identificateur), il suffit de récupérer la valeur associée à cette variable et la mettre dans la pile.

Une mauvaise manipulation de fichiers, de piles ou un mauvais calcul génèrent des erreurs. Ainsi, il est primordial d'intégrer le mécanisme des exceptions permettant de gérer les erreurs provoquées par de telles manipulations, par exemple :

– Une exception sera typiquement générée lorsqu'on tente une opération sur une pile vide (retirer un terme à partir d'une pile vide par exemple). Cette exception pourra être de type **NoSuchElementException** et on traitera le cas d'une pile vide, en affichant un message d'erreur approprié.

– Une exception de type **ArithmeticException** lorsqu'on effectue une division par 0 par exemple (auquel cas on affiche un message d'erreur approprié).

– Une exception de type **IOException** lorsqu'on ne réussit pas à lire ou écrire à partir du fichier (auquel cas on affiche un message d'erreur approprié).

– ….

d- Fournitures :

Les codes sources des interfaces sont mis à votre disposition sur la plate-forme **moodle**.

Vous devrez respecter intégralement les interfaces des modules. Vous y ajouterez vos propres méthodes, attributs ou les classes que vous jugerez utiles pour la réalisation de votre projet.

3. Travail à rendre et mini-soutenance

a- Rapport et tests :

Ce projet sera réalisé en binôme, ou éventuellement en monôme. A l'issue du projet, vous devrez rendre à votre enseignant de TP :

✔ Un rapport (au format pdf) résumant le travail effectué. Le contenu du rapport devra présenter le travail de conception et de programmation :

– Explication de la structure fonctionnelle de votre application (Diagramme d'héritage schématisant les relations qui existent entre les différentes classes et interfaces (Ne mettez pas les méthodes ni les attributs dans votre diagramme pour ne pas le surcharger).

– Présentation des différentes normes ou conventions adoptées (règles de nommage des classes, des variables, des packages,... ; normes d'organisation du code et les conventions relatives à sa lisibilité : indentation entête documentaire des méthodes, des classes,...).

– Justification des choix de conception orientée objet et d'implémentation. Un code inséré pour explication ou justification d'un choix donné, ne doit pas dépasser une demi-page et doit être abondamment commenté.
– Représentation des résultats (Jeu de tests valides avec expressions arithmétiques correctes et jeu de tests non valides avec expressions incorrectes). Il est important de tester votre projet à l'aide de jeux de tests significatifs. Ces derniers devront prouver le bon fonctionnement de votre application. Vous pouvez également joindre des classes de test.
– Dans la partie conclusion, discutez les limitations de votre application ainsi que les améliorations pouvant être effectuées.

✔ Codes sources et documentation : Les codes sources devront être clairs, lisibles, correctement indentés et intelligemment annotés. Pensez à insérer des entêtes documentaires des différentes méthodes et classes. Insérez également des commentaires pour clarifier une partie du code, une variable,.... Une archive compressée nommée **noms_GroupeTP.{tgz, tbz, zip, jar}** devra contenir l'ensemble des codes sources de votre projet ainsi que votre rapport.

b- Les Dates à retenir :

Les dates suivantes sont à retenir :

– **12 mai 2014 jusqu'à 18h :** Remise du code source et du rapport (Un espace sur la plate-forme **moodle** sera créé pour cet effet).
– **19 mai 2014 :** Mini-soutenance.

Le déroulement de la mini-soutenance est le suivant:
– Chaque étudiant sera questionné individuellement (Si le projet est réalisé en binôme, un effort important devra être fait pour mettre en avant et clairement, l'implication de chacun dans la réalisation du projet). L'étudiant en question aura 5 minutes pour présenter un aspect du projet réalisé. L'examinateur n'intervient pas pendant ce temps. Attention 5 minutes, c'est court, il vaut mieux avoir planifié un exposé.
– Ensuite l'examinateur aura 5 minutes pour juger le travail présenté et poser des questions.

Annexe 1 :

$java calc

Voulez-vous afficher la pile des calculs effectués ? (Oui/Non) : Non

Expression à calculer : 1 2 +
(1 + 2) = 3 avec { }

Expression à calculer : 1 a -
Expression de a : 13
(1 - 13) = -12 avec {a=13}

$java calc

Voulez-vous afficher la pile des calculs effectués ? (Oui/Non) : Oui

Expression à calculer : 3 2 * 1 -
Pile : [3.0]
Pile : [2.0, 3.0]
Pile : [6.0]
Pile : [1.0, 6.0]
Pile : [5.0]
((3 * 2) - 1) = 5 avec { }

Expression à calculer : 3 2 * -
Erreur : argument manquant.

Le fichier « Resultats.txt » contiendra :

Expression à calculer : 1 2 +
(1 + 2) = 3 avec { }

Expression à calculer : 1 a -
13
(1 - 13) = -12 avec {a=13}

Expression à calculer : 3 2 * 1 -
((3 * 2) - 1) = 5 avec { }

Le fichier « Erreurs.txt » contiendra :

Expression à calculer : 3 2 * -
Erreur : argument manquant.