UE INF1601
sujet de TP n^{o}2

L'objectif de ce TP est de réaliser un interpréteur d'expression en Notation Polonaise Inversée (NPI) à l'aide d'un AFD. Exemple :

- le calcul de $-12.4 + 5$ s'exprime en NPI par : `-12.4 5 +`
- le calcul de $-12 + (5.0 \times 3.5)$ s'obtient par la séquence : `-12 5.0 3.5 * +`
- le calcul de $-12 + 5 - -3.2 + 4$ s'obtient par la séquence : `-12 5 -3.2 4 + - +`

Première version

Les fonctionnalités attendues sont :

- la lecture des expressions sur l'entrée standard séparées par un `'\n'`
- l'affichage du résultat de l'expression sur la sortie standard à la rencontre d'un `'\n'`
- la gestion des nombres décimaux (sans exposant) signés (nombre négatif précédé d'un `"-"` collé au nombre)
- la gestion des opérations arithmétiques élémentaires : `"+" "-" "*" "/"`
- l'arrêt du programme sur la commande `'#'`
- la gestion de toutes les erreurs : **votre programme ne doit jamais planter**, comme une vraie calculatrice [1].

Contraintes à respecter :

- Il faut commencer par spécifier un AFD pour la reconnaissance des nombres et des opérateurs ; **commencez** par dessiner cet automate ! Vous devrez remettre un fichier au format pdf – pas de copie manuscrite scannée – contenant le schéma de votre automate avec votre fichier `NPI.java` ; pour information mon automate fait 9 états.
- L'automate ne sert qu'à reconnaître des unités lexicales, pas des expressions. C'est la gestion d'une pile dans les actions qui permet de vérifier la légalité des expressions.
- L'AFD doit être implémenté à l'aide de la classe [2] `afd.AFD` et en complétant le fichier `NPI.java`

1. En cas d'erreur dans une expression, l'AFD repart dans l'état initial.
2. la classe `afd.AFD` est fournie, mais doit éventuellement être adaptée à votre environnement système (regardez son code).

— Des actions doivent être ajoutées à l'AFD pour le calcul des expressions en utilisant une pile de double (`java.util.Stack<Double>`).
— Il est interdit d'utiliser les méthodes des classes String,Double et Integer pour la construction, la conversion et l'interprétation des nombres [3]. Plus précisément la construction des nombres sera réalisée de manière incrémentale à la lecture de chaque chiffre en se basant sur la décomposition suivante : $123 = (((1 \times 10) + 2) \times 10) + 3$

Seconde version

Il s'agit maintenant d'ajouter la possibilité d'utiliser des variables dans votre interpréteur d'expressions en NPI. Les identificateurs des variables sont limités à une ou plusieurs lettres majuscules. La définition d'une variable consiste à associer la valeur en sommet de pile à l'identificateur de la variable. Cette définition sera mémorisée dans une table. Toute définition est précédée d'un symbole >. En cas de redéfinition, la nouvelle valeur se substitue à l'ancienne. La référence à une variable (ie. son utilisation) consiste à remplacer l'identificateur d'une variable par la valeur calculée lors de sa définition.

Exemples :
— la mémorisation d'une valeur dans la variable PI : `3 0.14 + > PI`
— le calcul de $2 \times PI$ s'obtient par la séquence : `PI 2 *`

Travail à réaliser :

1. Modifiez graphiquement votre automate
2. Implémentez vos modifications dans `NPI.java`

3. Exemples de méthodes interdites : Integer.parseInt(String), Double.parseDouble(String), String.concat()...