TP 2 : INTERPRÉTEUR

INTRODUCTION

Pour le deuxième tp vous allez construire un logiciel qui lit une phrase. Cette phrase sera interprétée selon un de trois interpréteurs. Les deux premiers interpréteurs sont décrits dans cet énoncé. Le troisième vous sera donné une semaine avant la remise.

DESCRIPTION

Ce logiciel doit interpréter une phrase. Cette phrase sera lue dans un fichier texte. Elle est constituée d'une séquence de mots. Votre logiciel doit valider la syntaxe de la phrase. Ensuite, le logiciel demande à l'utilisateur de choisir l'interpréteur qui sera utilisé. Les interpréteurs doivent signaler out problème lors de l'interprétation. Ensuite, il offre la possibilité d'utiliser une autre interprétation. Finalement, le logiciel afficher l'état final de l'interpréteur.

ENTRÉES

Au lancement, votre logiciel doit demander à l'utilisateur d'entrer le nom du fichier qui contiendra la phrase à interpréter. S'il y a des problèmes à ouvrir le fichier, alors le logiciel doit afficher un message d'erreur. Lorsque le fichier est ouvert, le logiciel doit lire et mémoriser la phrase contenue dans le fichier. Cette phrase utilise la syntaxe suivante :

- Phrase :: séquence de Mot. Une phrase est une suite de 0 ou plus Mot. Chaque Mot de la phrase peut être entouré d'espace ' ', de tabulation '\t' et de fin de ligne '\n'.
- Mot :: un mot est une lettre minuscule. Les lettres de 'a' à 'g' sont utilisées. Les mots de 'a' à 'd' peuvent être suivi, optionnellement, par une valeur entière entre parenthèses. Il peut y avoir des espaces, tabulation et fin de ligne de chaque côté des parenthèses.

S'il y a une erreur de syntaxe dans la phrase, votre logiciel doit afficher un message d'erreur contenant le numéro de colonne et de ligne du caractère qui à une erreur. Le logiciel doit conserver l'information sur le numéro de ligne et de colonne pour chaque Mot. Ainsi, les erreurs lors de l'interprétation pourront aussi utiliser cette information.

Finalement, votre logiciel doit demander à l'utilisateur lequel des trois interpréteurs il doit utiliser. Pour cela, affichez un menu numérotant les trois interpréteurs. Aussi, placez un quatrième choix afin de terminer le logiciel.

```
Entrez le nom du fichier : test1.txt

1- Calculatrice.
2- Profil.
3- Mystère.
4- Fin du logiciel.

Entrez votre choix : 2
```

Quand l'interprétation choisie est terminée, votre logiciel revient à ce menu.

TRAITEMENT

Chaque interprétation contient une description de l'état interne de l'interpréteur (une liste de variable interne) et les sept actions associés à chaque Mot (partie spécifique). Aussi, tous les interpréteurs utilisent le même code pour l'interprétation d'une Phrase (partie générique). Voici la description de chaque interpréteur.

PREMIER INTERPRÉTEUR : CALCULATRICE

État interne :

- Accumulateur : un entier.
- Pile : une pile d'entiers.

Actions :

- a :
    1- Lire 2 valeurs sur la pile (pop).
    2- Additionner les deux valeurs.
    3- Placer le résultat sur la pile (push).
- a(v) : ajouter la valeur v au contenu de l'accumulateur.
- b :
    1- Lire 2 valeurs sur la pile (pop).
    2- Soustraire la première valeur à la deuxième.
    3- Placer le résultat sur la pile (push).
- b(v) : soustraire la valeur v au contenu de l'accumulateur.
- c :
    1- Lire 2 valeurs sur la pile (pop).
    2- Multiplier les deux valeurs.
    3- Placer le résultat sur la pile (push).
- a(v) : multiplier la valeur v avec contenu de l'accumulateur et placer le résultat dans l'accumulateur.
- d :
    1- Lire 2 valeurs sur la pile (pop).
    2- Diviser la première valeur par la deuxième.
    3- Placer le résultat sur la pile (push).
- a(v) : diviser l'accumulateur par v et placer le résultat dans l'accumulateur.
- e : Placer le contenu de l'accumulateur sur la pile (push).
- f : Lire une valeur sur la pile (pop) et la place dans l'accumulateur.
- g : Placer l'accumulateur à 0.

DEUXIÈME INTERPRÉTEUR : PROFIL

État interne :

- Nombre d'opérations(no) : un entier.
- Nombre d'accès à la pile(np) : un entier.
- Nombre d'accès à l'accumulateur(na) : en entier.

Actions :

- a, b, c, d :
  - np = np + 3
  - no = no + 1
- a(v), b(v), c(v), d(v) :
  - na = na + 2
  - no = no + 1
- e, f :
  - np = np + 1
  - na = na + 1
- g :
  - na = na + 1

CONSTRUCTION

Pour la construction du logiciel, vous allez devoir suivre une hiérarchie de classe spécifique. Cette hiérarchie permet une bonne représentation d'un interpréteur et va nous donner une bonne flexibilité pour adapter différents contextes pour l'interprétation.

REPRÉSENTATION DE L'INTERPRÉTEUR

Pour représenter les éléments interprétables (partie générale), nous allons utiliser une interface '`Expression`'. Les interpréteurs (partie spécialisée) vont utiliser l'interface '`ContexteInterpretation`'. L'interface pour les expressions va définir une méthode :

```
void interprete( ContexteInterpretation contexte )
```

Cette méthode va contenir le code générique pour l'interprétation. C'est ce code qui va faire appel au code spécialisé de l'interpréteur, défini par le contexte.

Ensuite vous devez définir une classe abstraite pour les mots du langage. Cette classe aura le nom '`Mot`' et implémentera la classe `Expression`. Chacun des 7 mots du langage sera représenté par une sous classe de '`Mot`' : `MotA`, `MotB`, `MotC`, `MotD`, `MotE`, `MotF` et `MotG`. Chacune de ces classes doit hériter de la classe `Mot`. Elles vont donc automatiquement implémenter l'interface '`Expression`'. Finalement, la classe '`Phrase`' va aussi implémenter l'interface `Expression`. Puisqu'elle contient une séquence de `Mot`, elle doit hériter d'un '`ArrayList< Mot >`', ce qui lui donnera toutes les méthodes d'un `ArrayList`.

CONTEXTE POUR L'INTERPRÉTATION

Afin de rendre l'interprétation facile à modifier, vous allez construire l'interface `ContexteInterpretation`.

```
public interface ContexteInterpretation {
	void a( MotA motA );
	void b( MotB motB );
	void c( MotC motC );
	void d( MotD motD );
	void e( MotE motE );
	void f( MotF motF );
	void g( MotG motG );
}
```

C'est cette interface qui sera implémentée par chaque interpréteur que vous voulez ajouter.

LIEN AVEC LE CODE

La méthode `interprete` dans les `Expression` de type `Mot` va simplement appeler les méthodes définies par le contexte d'interprétation reçu en argument. Pour la classe `Phrase`, la méthode `interprete` doit appeler l'interprétation sur chaque `Mot`, un après l'autre, en utilisant le même contexte d'interprétation pour chacune.

Ensuite, il vous reste à construire une classe implémentant le `ContexteInterpretation` pour chaque interpréteur. Cette classe va contenir les variables d'instance représentant l'état et va implémenter les méthodes pour chacune des actions.

Finalement, pour appeler votre interpréteur, il suffit de construire une instance de `Phrase` contenant des instances des `Mot`s représentant les mots lu dans le fichier. Vous construisez aussi une instance de la classe de contexte d'interprétation choisi par l'utilisateur. Finalement, il reste à démarrer l'interprétation en appelant la méthode `interprete` sur l'instance de `Phrase` avec le contexte d'interprétation en argument.

Il ne reste qu'à afficher le contenu de l'état final (définir un `toString` sur vos classes de contexte d'interprétation).

TRAITEMENT DES ERREURS

Lors de l'exécution, il peut y avoir des erreurs :

- Un pop sur une pile vide.
- Une division par zéro.

Si une de ces erreurs arrive, il faut afficher un message contenant le nom du mot qui a causé l'erreur avec le numéro de ligne et de colonne où il se trouvait dans le code original.

Toutes les erreurs dans le logiciel doivent être affichées sur le canal d'erreur (`System.err.println`) et être suivies d'un appel à `System.exit`.

DIRECTIVES

1. Le tp est à faire seul ou en équipe de deux.
2. Vous devez construire des classes appropriées.
3. Code :
    a. Vos méthodes ne devraient pas contenir plus de 5 instructions au total parmi : `if, for, while, try, switch.`
    b. Pas de `goto`, `continue`.
    c. Les `break` ne peuvent apparaître que dans les `switch`.
    d. Un seul `return` par méthode.
4. Indentez votre code.

COMMENTAIRES

- Commentez l'entête de chaque classe et méthode.
- Une ligne contient soit un commentaire, soit du code, pas les deux.
- Utilisez des noms d'identificateur significatif.
- Utilisez le français.
- Une ligne de commentaire ne devrait pas dépasser 80 caractères. Continuez sur la ligne suivante au besoin.
- Nous utilisons Javadoc :
    - La première ligne d'un commentaire doit contenir une description courte (1 phrase) de la méthode ou la classe.
        - Courte.
        - Complète.
        - Commencez la description avec un verbe.

        - Assurez-vous de ne pas simplement répéter le nom de la méthode, donnez plus d'information.
    - Ensuite, au besoin, une description détaillée de la méthode ou classe va suivre.
        - Indépendant du code. Les commentaires d'entêtes décrivent ce que la méthode fait, ils ne décrivent pas comment c'est fait.
        - Si vous avez besoin de mentionner l'objet courant, utilisez le mot 'this'.
    - Ensuite, avant de placer les **tags,** placez une ligne vide.
    - Placez les **tag** @param, @return et @throws au besoin.
        - @param : décris les valeurs acceptées pour la méthode.
    - Dans les commentaires, placer les noms de variable et autre ligne de code entre les tags <code> … </code>.
    - Écrivez les commentaires à la troisième personne.

REMISE

Remettre le tp par l'entremise de Moodle. Placez vos fichiers `'*.java'` dans un dossier compressé de Window, vous devez remettre l'archive (.zip). Le tp est à remettre avant le 16 novembre 23 :59.

ÉVALUATION

- Fonctionnalité (8 pts) : des tests partiels vous seront remis. Un test plus complet sera appliqué à votre tp.
- Structure (3 pt) : veillez à utiliser correctement le mécanisme d'héritage et de méthode.
- Lisibilité (2 pts) : commentaire, indentation et noms d'identificateur significatif.