TP 3, LISTE DOUBLEMENT CHAÎNÉE PERFORMANTE.

Présentation

Pour le dernier TP, vous allez modifier le code d'une liste doublement chaînée afin de diminuer le temps d'accès. Aussi, cette liste aura plusieurs types de chainons. Cette liste va contenir des indicateurs de localité. Une structure secondaire va servir d'index vers les indicateurs de localité. Cela va nous permettre d'accéder rapidement aux éléments au centre de la liste.

Premièrement, une description du travail est présentée. Ensuite, la technique pour les tests est décrite. Cela sera finalement suivi d'une section décrivant l'évaluation de votre TP et les modalités de remise.

Description

Vous allez trouver sur Moodle, dans la section *Type de données abstrait*, le fichier *Liste.java*. Ce fichier contient le code d'une liste doublement chaîné que vous pouvez utiliser comme code de départ. Votre programme devoir demander à l'utilisateur le nom d'un fichier. Ce fichier contient des valeurs qui seront placées dans une nouvelle liste. Ensuite, votre programme va demander des requêtes de recherche à l'utilisateur.

Description des données

Votre programme doit premièrement demander le nom d'un fichier à lire. Le fichier à lire va contenir une suite d'*élément* séparé par des *blancs*. Voici la description des trois chaînes de caractères pouvant être un *élément* :

- Une suite de caractères parmi : 'a'..'z', 'A'..'Z', '0'..'9', '_'. Cette suite de caractère représente une clef.
- Le caractère '{'.
- Le caractère '}'.

Le premier élément sera toujours une accolade ouvrante et le dernier élément sera toujours une accolade fermante. Les éléments sont séparés par un ou plusieurs *blancs*. Un blanc est un des trois caractères suivants :

- Espace : ' '.
- Tab : '\t'.
- Fin de ligne : '\n'.

Par exemple, le fichier pourrait contenir les données suivantes :

```
{
  { collection intervalle collection { add intervalle } }
  trouverPremierPossible { debut valeur debut size get debut getFin valeur { debut } debut }
  trouverDernierPossible { debut valeur début size get debut getDebut valeur { debut } debut }
  add { intervalle borneInferieur trouverPremierPossible intervalle getDebut borneSuperieur
  trouverDernierPossible borneInferieur intervalle getFin { i borneInferieur i borneSuperieur i
  intervalle remove borneInferieur union intervalley get } add borneInferieur intervalle }
  remove { intervalle borneInferieur trouverPremierPossible intervalle getDebut borneSuperieur
  trouverDernierPossible borneInferieur intervalley getFin i borneSuperieur borneInferieur i i
  { courant remove i i  borneSuperieur i borneInferieur {
  addAll i courant soustraction intervalle } } }
  appartient { v positionPossible  trouverPremierPossible v resultat { resultat get positionPossible
  appartient v } { e } resultat }
  taille { somme i { somme i taille } somme }
  union { contrainte resultat contrainte{ i resultat add i } resultat }
  intersection { contrainte resultat{ i1 { i2  contrainte resultat addAll i1 intersection i2 } }
  resultat }
  soustraction { contrainte resultat { i  contrainte resultat remove i } resultat }
}
```

Votre logiciel doit construire une liste contenant les éléments du fichier.

Description de la structure

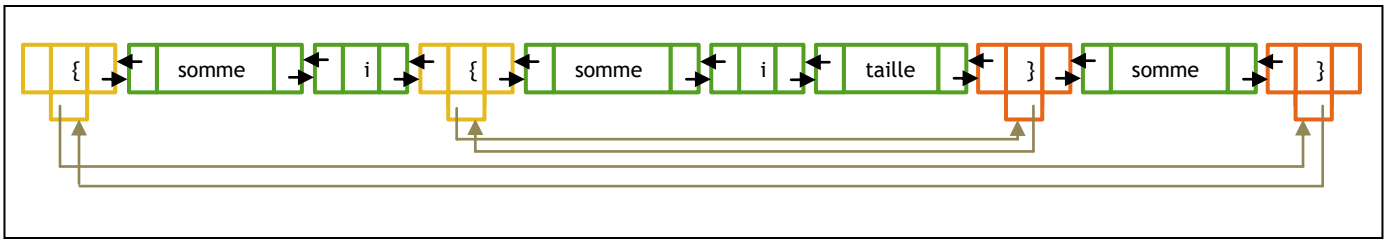
Vous allez devoir modifier le code de la liste doublement chaîné afin de lui ajouter des fonctionnalités. Chaque chainon de votre liste va contenir un élément, soit une chaîne de caractère, une accolade ouvrante ou une accolade fermante.

- Chaîne de caractère : chainon normal. Contiens le mot clef (String) et son indice (int). Contiens aussi les liens normaux vers le précédent et le suivant.
- Accolade ouvrante : chainon contenant les liens normaux vers le précédant et le suivant, mais en plus, il contient un lien vers l'accolade fermante qui lui équivaut (accolade associée).
- Accolade fermante : chainon contenant les liens normaux vers le précédant et le suivant, mais en plus, il contient un lien vers l'accolade ouvrante qui lui équivaut (accolade associée).

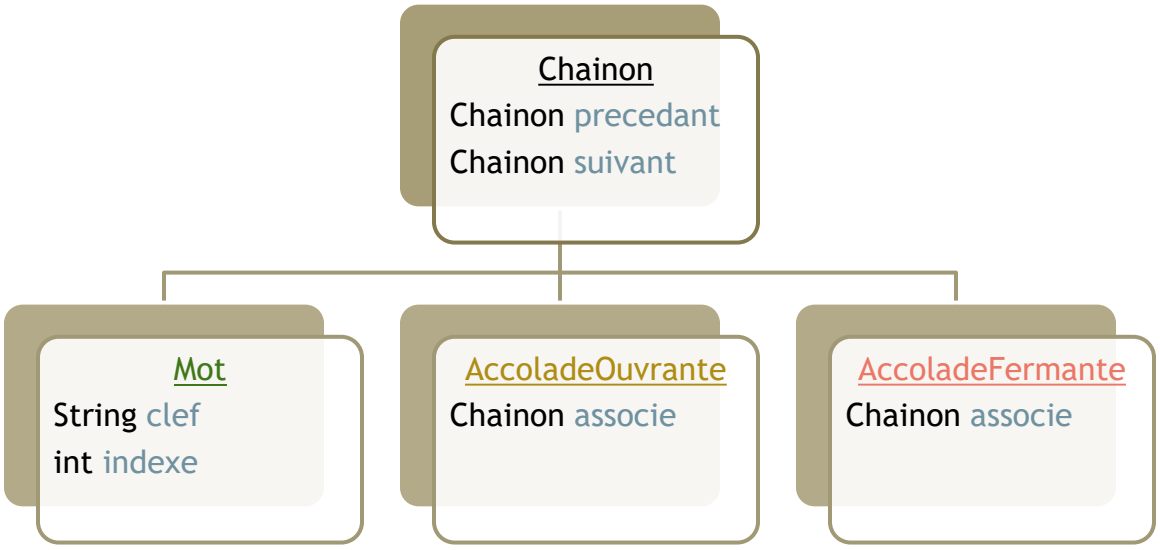
Par exemple, la ligne suivante :

```
{ somme i { somme i taille } somme }
```

Donnera la liste suivante :



Pour cela, vous devrez ajouter différentes classes pour représenter un Chainon. Ces classes devront hériter de la classe Chainon. Vous pouvez construire la hiérarchie qui vous convient. Voici une proposition pour cette hiérarchie, mais elle n'est pas obligatoire.



Cela va possiblement impliquer d'ajouter des méthodes d'insertion pour les chainons représentant les accolades. La méthode qui va ajouter une accolade fermante va devoir trouver l'accolade ouvrante qui lui est associée afin d'affecter les références d'une vers l'autre. Pour que cette opération soit performante, il suffit de construire une pile (`java.util.Stack`) d'`AccoladeOuvrante`. Toutes les fois qu'un `Chainon` représentant une accolade ouvrante est ajouté dans la liste, vous empilez une référence vers le `Chainon` ajouté. Toutes les fois qu'un `Chainon` représentant une accolade fermante est ajouté à la liste, vous lui associez le `Chainon` qui est au sommet de la pile et ensuite vous le dépilez.
Lorsque votre logiciel a terminé de charger et construire la liste contenue dans le fichier, il commence à interagir avec l'utilisateur.

Recherche rapide

Le logiciel demande des requêtes à l'utilisateur. Une requête peut commencer par la lettre 't' ou 'f'. Si la requête commence par 'f' (fin), alors le logiciel termine normalement. Si la requête commence par la lettre 't', alors cette lettre sera suivie par un espace et ensuite une clef (la cible de la recherche). Votre logiciel doit chercher la clef dans la liste. Cette méthode va conserver une

trace de la recherche pour accélérer la prochaine recherche. Aussi, cette méthode va affiche l'indexe de tous les chainons contenant une clef qui est comparée à la cible.

Pour accélérer la méthode de recherche, vous allez ajouter une variable d'instance à la classe Liste. Cette variable d'instance (dénoté par `p` dans ce texte) va contenir une pile (`java.util.Stack`) de Chainons qui conservera une référence sur les Chainons représentants des accolades ouvrantes qui ont été récemment visitées. Aussi, il faut ajouter une variable d'instance (dénoté par `d` dans ce texte) qui contiendra une référence sur le dernier Chainon qui a été recherché.

Première recherche

Lors du premier appel à l'algorithme de recherche, la pile `p` doit initialement être vide et la référence `d` doit initialement contenir la valeur `null`. Votre algorithme devra parcourir la liste des éléments un par un, à partir du premier élément jusqu'à ce que la cible cherchée soit trouvée. Lorsqu'un chainon représentant une accolade ouvrante est traversé, vous devez empiler la référence vers ce chainon sur la pile `p`. Lorsque le parcours de la liste traverse un chainon représentant une fermeture d'accolade, vous devez dépiler le sommet de la pile `p`. Lorsque le chainon contenant la cible est trouvé, vous devez placer une référence vers ce chainon dans la variable `d`. Si toute la liste est parcourue sans trouver la cible, alors la variable `d` reste à `null`.

Tant que la variable `d` reste à `null`, alors vous considérez que c'est la première recherche.

Recherche subséquente

Pour les recherches suivantes, lorsque `d` n'est plus à `null`, alors les recherches ne commencent plus au début de la liste. Une recherche commence par comparer avec le dernier élément trouvé (variable `d`).

Ensuite, si la valeur n'est pas trouvée, l'algorithme doit commencer la recherche (le parcours) à partir du chainon qui est au sommet de la pile `p`. Les éléments sont parcourus un par un, jusqu'à ce que la cible cherchée soit trouvée ou que vous trouviez l'accolade fermante équivalente à l'accolade ouvrante au sommet de la pile `p`. (N'oubliez pas d'empiler toute référence vers des accolades ouvrantes que vous traversez en chemin et de les dépiler lorsque vous traversez une accolade fermante.) Si vous trouvez l'accolade fermante associée à l'accolade ouvrante au début du parcours, alors vous devez dépiler la référence à l'accolade ouvrante et recommencer un nouveau parcours à partir de la nouvelle référence située au sommet de la pile `p`.

Pour une même recherche, lorsque vous parcourez l'intervalle entre deux accolades, si vous rencontrez un sous-intervalle d'accolade qui a déjà été parcouru dans cette recherche, alors vous devez sauter par-dessus cet intervalle. Pour faire ce saut, vous devez simplement suivre la référence vers l'accolade associée.

Affichage

Lors d'une recherche, votre logiciel doit afficher, sur une seule ligne, chaque clef qui a été comparée avec la cible. Cet affichage doit avoir lieu dans l'ordre que le programme rencontre les clefs.

Élément d'évaluation

Directives

1. Le TP est à faire seul ou en équipe de deux.
2. Code :
    a. Pas de `goto`, `continue`.
    b. Les `break` ne peuvent apparaitre que dans les `switch`.
    c. Un seul `return` par méthode.
3. Indentez votre code. Assurez-vous que l'indentation est faite avec des espaces.
4. 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.
    - 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 … }.
        - Écrivez les commentaires à la troisième personne.

Remise

Remettre le TP par l'entremise de Moodle. Placez vos fichiers `'*.java'` dans un dossier compressé (**zip**) de **Windows**, vous devez remettre l'archive. Le TP est à remettre avant le 16 décembre 23:55.

Évaluation

- Fonctionnalité (9 pts) : des tests vont être disponibles sur Moodle.
- Structure (2 pts) : veillez à utiliser correctement le mécanisme d'héritage et de méthode.
- Lisibilité (2 pts) : commentaire, indentation et noms d'identificateur significatif.