Devoir 2 : Listes doublement chaînées

IFT1025

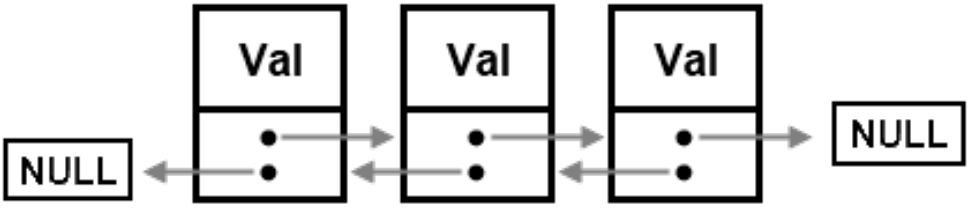
10 mars 2021

1 Mise en contexte

Les listes doublement chaînées sont des structures de données semblables aux listes simplement chaînées. L'allocation de la mémoire est faite au moment de l'exécution. En revanche, par rapport aux listes simplement chaînées, la liaison entre les éléments se fait grâce à deux pointeurs, un pointant vers l'élément précédent, l'autre vers l'élément suivant.

La premier élément de la liste se nomme la tête, et le pointeur précédent de la tête doit pointer vers NULL. Pour ce qui est du dernier élément, il se nomme la queue, pour lequel son pointeur suivant pointe aussi vers NULL. Pour accéder à un élément spécifique contenu dans la liste, celle-ci doit être parcourue de la tête vers la queue, ou bien l'inverse.

Les listes doublements chaînées sont couramment utilisés dans divers types d'applications, comme les lecteurs de musique qui possèdent un bouton suivant et un bouton précédent pour naviguer dans une liste de lecture, ou bien aux navigateurs Web qui ont une cache qui sauvegarde l'historique de navigation permettant ensuite de naviguer à travers cette historique vers l'arrière ou vers l'avant.



2 Problème

Pour le deuxième devoir, on vous demande d'implémenter une liste doublement chaînée, sans bien-sûr utiliser celles qui vous sont fournies par JAVA. Votre liste doit être *générique*. Elle doit donc pouvoir contenir différents types de données en même temps.

Le fonctionnement du programme ressemble sensiblement à celui du Devoir 1. Un fichier en entrée vous sera fourni en exemple, de même qu'un fichier de sortie, pour vous aider lors de votre développement. Ce fichier comprendra un sens de tri *asc* (croissant) ou *desc* (décroissant) ainsi que les éléments de la liste, devant être ajoutés à votre instance. Cependant, contrairement aux listes doublement chaînées de JAVA, les éléments contenus dans votre liste doivent toujours être triés en ordre croissant ou décroissant, selon le paramètre fourni dans le fichier en entrée présenté précédemment. Les exemples de la page suivante vous aideront à comprendre.

Pour comparer vos éléments de différents types, utilisez la classe *Comparable* et sa fonction *compareTo* ainsi que le concept de *génériques* que vous avez vu en cours pour implémenter de votre classe Item.

Ce travail vous permet de mettre en pratique des concepts importants des structures de données tout en travaillant avec les génériques et les interfaces!

2.1 Contenu de votre programme

- Une classe Item, la classe générique pouvant contenir des éléments de différents types *Comparables<T>*. La classe item doit *override* la définition de la méthode *compareTo* provenant de la classe *Comparable*.
- Une classe Noeud, la classe qui contiendra un Item ainsi que le pointeur précédent ainsi que le pointeur suivant vers les noeuds concernés.
- Une classe ListeDoublementChainee, la classe principale s'occupant de l'initialisation de la liste doublement chaînée ainsi que l'ajout de noeuds à celle-ci. Cette classe offre aussi une méthode pour écrire les éléments nécessaires dans le fichier de sortie.

2.2 Entrées et sorties

Une coquille de programme vous est déjà fournie pour lire le fichier input.txt. De plus, le fichier *Main.java*, aussi fourni, ne devrait pas être modifié, quoi qu'il ne vous est pas interdit de le faire. C'est aussi celui-ci qui doit être exécuté pour lancer le programme. Pour écrire dans le fichier de sortie, vous pouvez faire comme au devoir 1, soit diriger les sorties du programme avec l'opérateur >, ou bien dynamiquement créer un fichier et écrire à l'intérieur de celui-ci par programmation. Si vous procédez ainsi, n'oubliez pas de transformer vos System.out.prinln(). Plusieurs exemples sont disponibles sur internet.

Voici des exemples d'entrées et de sorties que vous devrez traiter.

```
asc [1,2,3]
desc [3,2,1]
asc [2,1,C,a]
desc [2,1,C,a]
asc [1,5,7,a,h,r,b,j,0.4,0.9,5.90,100]
desc [1,5,7,a,h,r,b,j,0.4,0.9,5.90,100]
```

Listing 1: Exemple d'entrée dans le fichier input.txt

```
----------
Sens du tri: asc
Liste originale: [1,2,3]
Noeuds du debut vers la fin: 1->2->3
Noeuds de la fin vers le debut: 3->2->1
----------
----------
Sens du tri: desc
Liste originale: [3,2,1]
Noeuds du debut vers la fin: 3->2->1
Noeuds de la fin vers le debut: 1->2->3
----------
----------
Sens du tri: asc
Liste originale: [2,1,C,a]
Noeuds du debut vers la fin: 1->2->C->a
Noeuds de la fin vers le debut: a->C->2->1
----------
----------
Sens du tri: desc
Liste originale: [2,1,C,a]
Noeuds du debut vers la fin: a->C->2->1
Noeuds de la fin vers le debut: 1->2->C->a
----------
----------
Sens du tri: asc
Liste originale: [1,5,7,a,h,r,b,j,0.4,0.9,5.90,100]
Noeuds du debut vers la fin: 0.4->0.9->1->5->5.9->7->100->a->b->h->j->r
Noeuds de la fin vers le debut: r->j->h->b->a->100->7->5.9->5->1->0.9->0.4
----------
----------
Sens du tri: desc
Liste originale: [1,5,7,a,h,r,b,j,0.4,0.9,5.90,100]
Noeuds du debut vers la fin: r->j->h->b->a->100->7->5.9->5->1->0.9->0.4
Noeuds de la fin vers le debut: 0.4->0.9->1->5->5.9->7->100->a->b->h->j->r
----------
```

Listing 2: Sortie attendue dans le fichier output.txt

2.3 Cas spéciaux et indices

- Comme mentionné plus tôt, votre classe Item doit être générique et pouvoir contenir des éléments de plusieurs types. Dans ce travail, les trois types avec lesquels vous travaillerez sont les *entiers* (Integer), les *doubles* (Double) et les *strings* (String). Vous devez donc, lors de la création d'instances de votre classe Item, déterminer le type de l'élément lu dans le fichier d'entrée pour ensuite créer votre Item avec le bon type (Integer, Double ou String). Deux fonctions utilitaires vous sont fournies dans le fichier Main.java qui vous permettront, pour une string *s*, de vérifier si son contenu peut être *cast* en *Integer* ou en *Double* Si l'élément lu est une *String*, vous n'avez donc pas de conversion à faire.

- Vous pouvez remarquer dans le fichier en entrée que certaines listes contiennent des *strings* avec *entiers* et des *doubles*. Dans votre méthode *compareTo*, vous devez donc identifier ce cas spécila et, puisqu'il n'est pas possible de *cast* un *entier* en *string*, de convertir l'entier en strings avec la méthode *toString()* et d'ainsi appeler la méthode *compareTo* sur deux éléments de même type. De plus, la même situation va arriver lorsque vous tenterez de comparer un *Integer* avec un *Double*, ou l'inverse, et vous obtiendrez alors cette erreur: *Exception in thread "main" java.lang.ClassCastException: class java.lang.Integer cannot be cast to class java.lang.Double.* Vous devez donc ajoutez un cas particulier dans votre méthode *compareTo* qui s'occupe de convertir à la main votre *entier* en *double*, et d'ensuite retourner le résultat de la vérification. Effectuez toujours la conversion du moins précis vers le plus précis, pour ne justement pas perdre de précision.

- L'interface de la classe ListeDoublementChainee (*IListeDoublementChainee.java*) vous est fournie. Vous n'avez qu'à l'implémenter.

- Les lettres majuscules dans une *String* sont toujours inférieures aux lettres minuscules. En d'autres mots, "Z" est inférieur à "a". Jettez un coup d'oeil à l'exemple présenté plus tôt.

- Dans le fichier de sortie, l'impression de la ligne "Noeuds de la fin vers le début" ne correspond pas seulement à imprimer la ligne "Noeuds du debut vers la fin" à l'envers. Vous devez, dans votre liste doublement chainée, imprimer le dernier noeud (la tête), son précédent, et ainsi de suite.

3 Évaluation

- 25% des points si votre programme compile sans erreur
- 25% des points si votre programme exécute sans erreur.
- 25% des points si votre programme produit les bons résultats
- 25% des points pour la clarté de votre code, l'utilisation du camelCase et le respect des consignes.
- Une pénalité de 10% par jour de retard sera appliquée dès la première seconde de chaque période de 24h, en commençant à 00h00 le 28 mars 2021.

4 Questions

Un forum de formation d'équipes et un forum de discussion sera créé pour le Devoir 2. Vous pouvez aussi envoyez vos questions à antoine.st-laurent@umontreal.ca.

Bon travail!