Travail pratique #2

Ce travail doit être fait individuellement.

Notions mises en pratique : respect d'un ensemble de spécifications, implémentation d'une interface (définissant une TDA), généricité, et utilisation d'une liste chainée.

1. Description générale

1.1 DÉFINITION DU TDA LISTE TRIÉE

L'interface `IListeTriee` définit le TDA liste triée. Lisez attentivement la définition de ce TDA dans la Javadoc de l'interface `IListeTriee`. Ce travail consiste à implémenter ce TDA à l'aide d'une liste chainée.

2. Implémentation

2.1 DÉTAILS ET CONTRAINTES D'IMPLÉMENTATION

Votre travail consiste à implémenter toutes les méthodes de l'interface `IListeTriee` (fournie) dans une classe nommée `ListeTrieeChainee`, en utilisant une liste chainée comme structure de données sous-jacente qui contiendra les éléments de la liste. Voici les contraintes d'implémentation à respecter pour définir cette classe.

2.1.1 Implémentation de la classe `ListeTrieeChainee` : structure de données

Votre liste triée doit être représentée en mémoire par une liste chainée (liste de maillons chainés), sans tête ni remorque. Les éléments, stockés dans chaque maillon de la liste, sont de type T, où T doit implémenter l'interface `java.lang.Comparable`. Cette interface ne contient qu'une seule méthode, la méthode `compareTo`. Vous connaissez déjà cette méthode, car elle existe dans la classe `String`, qui implémente effectivement l'interface `Comparable`. La méthode `compareTo` permet de déterminer si un élément est plus petit, égal ou supérieur à un autre élément (voir Javadoc de l'interface `Comparable` pour comprendre comment utiliser cette méthode). L'implémentation de cette interface par le type `T` des éléments d'une liste triée assure donc la présence d'une méthode permettant d'ordonner les éléments entre eux, selon leur ordre de grandeur, qui est nécessaire pour maintenir une liste triée. Pour imposer cette contrainte sur le type `T`, l'entête de votre classe doit être écrit comme ceci :

```
public class ListeTrieeChainee <T extends Comparable> implements IListeTriee <T>
```

VOUS DEVEZ RESPECTER CET ENTÊTE.

De plus, comme cette liste triée est une liste non indicée, elle doit maintenir une position courante (sauf lorsqu'elle est vide). Pour ce faire, vous devez utiliser un attribut de type `Maillon<T>`, qui pointe vers le maillon contenant l'élément courant. Lorsque la liste ne contient aucun élément, cet attribut doit être initialisé à `null`, pour signifier qu'il n'y a aucune position courante (et donc, aucun élément courant).

Votre classe `ListeTrieeChainee` doit contenir **les trois attributs d'instance suivants** :

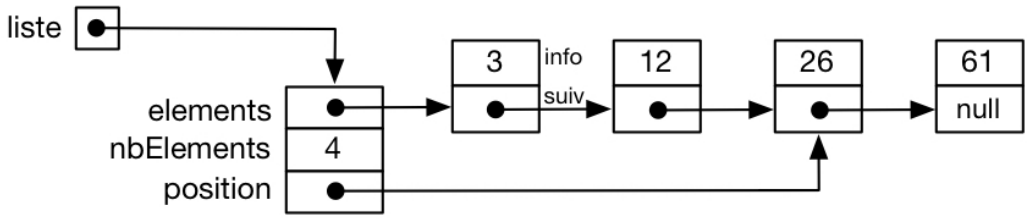
Nom attribut	Type	Description
`elements`	`Maillon<T>`	Premier maillon de la **liste** chainée utilisée pour représenter cette liste triée en mémoire. Lorsque cet attribut est `null`, c'est que cette liste triée est vide.
`nbElements`	`int`	Représente la **taille** de cette liste triée (le nombre d'éléments qu'elle contient). La valeur de cet attribut est toujours supérieure ou égale à 0. *N'oubliez pas d'ajuster cette valeur lors de l'ajout / la suppression d'éléments.*
`position`	`Maillon<T>`	Maillon qui représente la position courante de cette liste triée, et qui pointe toujours sur le maillon qui contient l'élément courant. Cet attribut est `null` seulement lorsque cette liste triée est vide.

Mélanie Lord
lord.melanie@uqam.ca

NOTES :

- ***AUCUN autre attribut d'instance*** *n'est permis. Cette structure de données sera testée, et vous pouvez perdre plusieurs points aux tests si elle n'est pas respectée.*
- ***Il est important de respecter le nom, et le type de ces attributs****, sinon les tests ne compileront pas et vous perdrez des points.*

Exemple d'une représentation en mémoire d'un objet (`liste`) de type `ListeTrieeChainee<Integer>` :



La figure ci-dessus montre une liste triée chainée contenant 4 éléments, et dont l'élément courant (celui à la position courante) est 26.

2.1.2 Implémentation de la classe `ListeTrieeChainee` : constructeur

Votre classe doit posséder un seul constructeur, sans argument, qui crée une liste triée vide (qui ne contient aucun élément).

2.1.3 Implémentation de la classe `ListeTrieeChainee` : méthodes d'instance publiques

Comme la classe `ListeTrieeChainee` implémente l'interface `IListeTriee` (fournie avec cet énoncé du TP), elle doit donc implémenter toutes les méthodes listées dans cette interface, en plus de redéfinir la méthode `toString`. NOTEZ que la méthode `toString` vous est fournie, et vous devez simplement la copier dans votre classe.

NOTES :

- ***Lisez bien la Javadoc*** *de chaque méthode dans l'interface* `IListeTriee` *pour bien comprendre et respecter ce qu'elles doivent faire.*
- ***Les seules méthodes publiques*** *pouvant se trouver dans la classe* `ListeTrieeChainee` ***sont celles listées dans l'interface*** `IListeTriee` *à l'exception de la redéfinition de la méthode* `toString` *dont l'implémentation vous est fournie avec l'énoncé de ce TP (voir fichier toString.txt). Toute autre méthode doit être privée.*

Il est à noter que lorsqu'une méthode à implémenter doit vérifier l'égalité entre des éléments de types `T`, la méthode `boolean equals (Object o)` du type `T` peut être utilisée (au lieu de la méthode `compareTo`).

2.2 TESTS DE VOTRE CLASSE LISTETRIEECHAINEE

Vous n'avez pas d'application à remettre, cependant, vous devrez coder des programmes / méthodes pour tester votre classe `ListeTrieeChainee`. Comme les éléments contenus dans une liste triée doivent implémenter l'interface `Comparable`, vous pouvez utiliser la classe `String` ou la classe `Integer` (comme type d'éléments dans votre liste) qui implémentent cette interface en plus de redéfinir la méthode `equals`. Vous pourriez donc instancier une liste triée contenant ce type d'éléments pour faire vos tests.

3. Interface, classes, et fichiers fourni(e)s (**À UTILISER et à NE PAS modifier**)

Interface `IListeTriee`	Interface à implémenter pour ce travail.
Classe `Maillon`	Classe utilisée pour la création de listes chainées (représente un maillon de la liste).
Classe `ListeVideException`	Classe d'exception implicite utilisée dans certaines méthodes de `IListeTriee`.
Fichier *toString.txt*	Fichier qui contient la méthode `toString` **que vous devez copier-coller dans votre classe** `ListeTrieeChainee`.
Classe `TestsPartielsListeTriee`	Classe de tests PARTIELS pour tester PARTIELLEMENT votre classe `ListeTrieeChainee`. Vous devrez ajouter des tests pour tester correctement votre implémentation.

IMPORTANT : Certains des éléments fournis seront utilisés tels quels dans les tests de votre programme. **Il est donc important de ne pas les modifier**.

4. Précisions

- **VOUS DE DEVEZ PAS utiliser un algorithme de tri** pour trier la liste. Il faut plutôt, lors de l'ajout d'un élément, toujours l'ajouter à la bonne place pour constamment maintenir le tri.
- Vous devez respecter le **principe d'encapsulation** des données.
- Vous NE DEVEZ PAS utiliser la classe `ArrayList` (ou tout autre type de collections) ni les tableaux. Les seules classes/interfaces permises sont : toutes les classes d'exception requises, `Maillon`, `IListeTriee`, `ListeTrieeChainee`, et `String` (seulement pour la méthode `toString` fournie).
- N'oubliez pas d'écrire la **Javadoc** (pour TOUTES les méthodes).
- **Aucune variable globale** (autre que les attributs d'instance mentionnés à la section 2.1.1) n'est permise. Vous pouvez cependant ajouter des constantes globales (`final static`) si vous le jugez pertinent.
- Réutilisez votre code autant que possible. Vous pouvez (et devriez) faire des **méthodes privées** pour bien structurer/modulariser votre code (séparation fonctionnelle). ATTENTION : toute méthode qui n'est pas dans l'interface `IListeTriee` (autre que `toString`) doit être privée (`private`).
- Il ne doit y avoir aucun affichage dans vos méthodes (pas de `System.out`)
- Votre classe doit se trouver dans le **paquetage par défaut**.
- Vous DEVEZ respecter le style Java.
- Vos fichiers à remettre doivent être encodés en UTF8.
- Votre code doit compiler et s'exécuter avec le JDK 8.

Le non-respect de toute spécification ou consigne se trouvant dans l'énoncé du TP est susceptible d'engendrer une perte de points.

NOTE : Si quelque chose est ambigu, obscure, s'il manque de l'information, si vous ne comprenez pas les spécifications, si vous avez des doutes… vous avez la responsabilité de vous informer auprès de votre enseignante.

5. Détails sur la correction

5.1 LA QUALITÉ DU CODE (30 POINTS)

Concernant les critères de correction du code, lisez attentivement le document “Critères généraux de correction du code Java". De plus, votre code sera noté sur le respect des conventions de style Java vues en classe (dont un résumé se trouve dans le document "Conventions de style Java"). **Ces deux documents peuvent être téléchargés dans la section TRAVAUX PRATIQUES (ET BOITES DE REMISE) sur la page Moodle du cours**.

Note : Votre code sera corrigé sur la totalité ou une partie des critères de correction indiqués ci-dessus, ainsi que sur le respect des spécifications / consignes mentionnées dans ce document.

5.2 L’EXÉCUTION (70 POINTS)

Un travail qui ne compile pas se verra attribuer la note 0 pour l’exécution.

Votre code sera testé en tout ou en partie. Les points seront calculés sur les tests effectués. Ceci signifie que si votre code fonctionne dans un cas x et que ce cas x n’est pas testé, vous n’obtiendrez pas de points pour ce cas. Il est donc dans votre intérêt de vous assurer du bon fonctionnement de votre programme dans tous les cas possible.

Une classe de tests PARTIELS vous est fournie (servez-vous-en), mais vous devrez aussi composer vos propres tests pour tester correctement votre classe. Les tests qui seront exécutés pour la correction seront différents de ceux fournis.

6. Date et modalités de remise

6.1 REMISE

Date de remise : Au plus tard le **25 mars 2022** à 23h59.
Le fichier à remettre : `ListeTrieeChainee.java` (**PAS** dans une archive zip, rar, etc.)

Remise via Moodle uniquement.

Vous devez remettre (téléverser) votre fichier sur le site du cours (Moodle). Vous trouverez la boîte de remise dans la section **TRAVAUX PRATIQUES (ET BOITES DE REMISE)**.

VÉRIFIEZ BIEN QUE VOUS AVEZ REMIS LE BON FICHIER SUR MOODLE (le .java et non le .class, par exemple).

6.2 POLITIQUE CONCERNANT LES RETARDS

Une pénalité de 10% de la note finale, par jour de retard, sera appliquée aux travaux remis après la date limite. La formule suivante sera utilisée pour calculer la pénalité pour les retards : `Nbr points de pénalité = m / 144`, où `m` est le nombre de minutes de retard par rapport à l'heure de remise. Ceci donne 10 points de pénalité pour 24 heures de retard, 1.25 point de pénalité pour 3 heures, etc.

Aucun travail ne sera accepté après 1 jour (24 h) de retard, et la note attribuée sera 0.

6.3 REMARQUES GÉNÉRALES

- Aucun fichier reçu par courriel ne sera accepté. **En d'autres termes, un travail reçu par courriel sera considéré comme non remis.**
- Vous avez la responsabilité de conserver des copies de sauvegarde de votre travail (sur disque externe, Moodle, Dropbox, Google Drive, One Drive, etc.). La perte d'un travail due à un vol, un accident, un bris… n'est pas une raison valable pour obtenir une extension pour la remise de votre travail.
- **N'oubliez pas d'écrire (entre autres) votre nom complet, et votre code permanent dans l'entête de la classe à remettre.**
- **N'attendez pas à la dernière minute pour commencer le travail, vous allez fort probablement rencontrer des problèmes inattendus!**

Ce travail est strictement individuel. Le règlement sur le plagiat sera appliqué sans exception. Vous devez ainsi vous assurer de ne pas échanger du code avec des collègues ni de laisser sans surveillance votre travail au laboratoire. Vous devez également récupérer rapidement toutes vos impressions de programme au laboratoire.

BON TRAVAIL !