Travail pratique #2

Ce travail doit être fait individuellement.

Notions mises en pratique : le respect d'un ensemble de spécifications, l'implémentation d'une interface définissant un TDA, la généricité, et l'utilisation de listes chainées.

Prenez le temps de lire cet énoncé au complet AVANT de commencer votre TP.

1. Description générale

1.1 Définition DU TDA File de priorité

Dans le cadre de ce TP, nous dirons qu'une file de priorité est une file dont chacun des éléments possède une valeur de priorité représentée par un nombre entier. Plus la valeur de la priorité d'un élément est grande, plus cet élément est dit de priorité élevée. Dans ce type de file de priorité, la priorité d'un élément est donc déterminée par deux facteurs :

1) Le moment d'entrée dans la file : premier arrivé, premier sorti (comme une file ordinaire)
2) La valeur de la priorité de l'élément.

Les éléments dans cette file sont ordonnés selon la valeur de leur priorité (de la plus grande à la plus petite). De plus, lorsque plusieurs éléments de même priorité se trouvent dans la file, ceux-ci sont ordonnés selon leur ordre d'arrivée dans la file (premier arrivé, premier sorti). L'élément en tête de cette file de priorité est donc toujours le premier élément arrivé dans la file parmi ceux qui ont la valeur de priorité la plus élevée.

Supposons qu'on représente un élément de cette file comme ceci : `e(p)` où `e` représente un élément quelconque, et le nombre `p` entre parenthèses est la valeur de priorité de cet élément. Voici à quoi pourrait ressembler une telle file de priorité :

tête [e1(10), e2(10), e3(10), e4(7), e5(6), e6(6), e7(1), e8(1), e9(1), e10(1)] fin

On peut voir ce type de file de priorité comme une suite de files internes, ordonnées selon la priorité de leurs éléments (la plus grande priorité en tête de file et la plus petite priorité en fin de file). Par exemple, dans la file de priorité illustrée ci-dessus, la première file interne (bleue) contient 3 éléments de priorité 10, dont la tête est `e1(10)` et la fin `e3(10)`, la seconde file interne (verte) contient un seul élément de priorité 7, `e4(7)`, la troisième file interne (orange) contient deux éléments de priorité 6 dont l'élément `e5(6)` est la tête et `e6(6)` est la fin, et finalement, la dernière file interne (rouge) contient 4 éléments de priorité 1, dont la tête est l'élément `e7(1)` puis la fin est l'élément `e10(1)`. Si l'on considère seulement une file interne, les éléments y sont ordonnés selon le principe FIFO (*first in first out*). Cela signifie que, pour la file interne bleue, par exemple (dont les éléments sont de priorité 10), le premier élément qui est arrivé dans cette file interne est `e1(10)`, suivi de `e2(10)`, suivi de `e3(10)`.

On dira que l'élément le plus prioritaire dans une file de priorité est l'élément en tête de la file : le premier élément arrivé dans la file parmi ceux qui sont de la plus grande priorité. Par exemple, dans la file de priorité illustrée ci-dessus, l'élément le plus prioritaire est `e1(10)`. Si celui-ci est défilé alors l'élément le plus prioritaire deviendra `e2(10)`, etc. On dira aussi que l'élément le plus prioritaire d'une priorité donnée est l'élément en tête de la file interne de cette priorité. Par exemple, dans la file de priorité illustrée ci-dessus, l'élément le plus prioritaire de priorité 1 est `e7(1)`.

Enfilement d'un élément

Lorsqu'on enfile un élément :

1) S'il existe déjà, dans la file de priorité, au moins un élément de même priorité que celle de l'élément à enfiler, on enfile l'élément à la fin de la file interne contenant les éléments de cette priorité.
2) S'il n'existe pas, dans la file de priorité, d'éléments de même priorité que celle de l'élément à enfiler, l'élément est alors inséré en respectant l'ordre (décroissant) des priorités des éléments et devient la tête d'une nouvelle file interne.

Exemples :

Soit la file de priorité vide suivante : `tête [] fin`

Après l'enfilement de `e1(3)` : `tête [ **e1(3)** ] fin`
Après l'enfilement de `e2(7)` : `tête [ **e2(7),** e1(3) ] fin`
Après l'enfilement de `e3(7)` : `tête [ e2(7), **e3(7),** e1(3) ] fin`
Après l'enfilement de `e4(5)` : `tête [ e2(7), e3(7), **e4(5),** e1(3) ] fin`
Après l'enfilement de `e5(7)` : `tête [ e2(7), e3(7), **e5(7),** e4(5), e1(3) ] fin`
Après l'enfilement de `e6(9)` : `tête [ **e6(9),** e2(7), e3(7), e5(7), e4(5), e1(3) ] fin`
Après l'enfilement de `e7(5)` : `tête [ e6(9), e2(7), e3(7), e5(7), e4(5), **e7(5),** e1(3) ] fin`
Après l'enfilement de `e8(3)` : `tête [ e6(9), e2(7), e3(7), e5(7), e4(5), e7(5), e1(3), **e8(3)** ] fin`
Etc.

Défilement d'un élément

Dans ce genre de file de priorité, il existe deux types de défilement.

1) Défilement de l'élément le plus prioritaire (celui en tête de la file de priorité).

   Exemples :

   Soit la file de priorité suivante : `tête [ e2(7), e3(7), e4(5), e1(3) ] fin`
   Un premier défilement défile `e2(7)` : `tête [ e3(7), e4(5), e1(3) ] fin`
   Un second défilement défile `e3(7)` : `tête [ e4(5), e1(3) ] fin`
   Etc.

2) Défilement de l'élément le plus prioritaire d'une priorité donnée (élément en tête de la file interne contenant les éléments de cette priorité) :

   Exemples :

   Soit la file de priorité suivante :
   `tête [ e6(9), e2(7), e3(7), e5(7), e4(5), e7(5), e1(3) ] fin`

   Un défilement de priorité 7 défile `e2(7)` :
   `tête [ e6(9), e3(7), e5(7), e4(5), e7(5), e1(3) ] fin`

   Un second défilement de priorité 7 défile `e3(7)` :
   `tête [ e6(9), e5(7), e4(5), e7(5), e1(3) ] fin`

   Un défilement de priorité 3 défile `e1(3)` :
   `tête [ e6(9), e5(7), e4(5), e7(5) ] fin`

   Un défilement de priorité 5 défile `e4(5)` :
   `tête [ e6(9), e5(7), e7(5) ] fin`
   Etc.

2. Implémentation de la classe FilePrioChainee

2.1 Détails et contraintes d'Implémentation

L'interface `IFilePrio` (fournie avec l'énoncé du TP) définit le TDA file de priorité décrit à la section précédente. Votre travail consiste à implémenter toutes les méthodes de cette interface dans une classe nommée **`FilePrioChainee`**, en utilisant une liste de maillons chainés comme structure de données sous-jacente pour stocker en mémoire les éléments de cette file de priorité. Voici les contraintes d'implémentation à respecter pour définir cette classe.

2.1.1 STRUCTURE DE DONNÉES ET TYPE GÉNÉRIQUE

Une file de priorité telle que décrite dans la section 1 peut être représentée en mémoire par **une liste de maillons chainés**. C'est le premier maillon de cette liste qui est la tête de la file de priorité. Vous devez utiliser la classe `Maillon` (fournie avec l'énoncé de ce TP) pour représenter chacun des maillons de la chaine. Chacun des maillons de cette liste contient un élément (une info) de type `T`. De plus, comme c'est une file de priorité, `T` doit implémenter l'interface `ITachePrio` fournie avec l'énoncé de ce TP. L'implémentation de cette interface assure la présence de méthodes pour obtenir ou modifier la priorité des éléments contenus dans cette file de priorité. Donc, l'entête de votre classe DOIT ËTRE écrit EXACTEMENT comme ceci :

```
public class FilePrioChainee<T extends ITachePrio> implements IFilePrio<T> {...}
```

***Note** : `<T extends ItachePrio>` signifie que le type `T` doit implémenter l'interface `ItachePrio`.*

ATTENTION : Votre classe doit absolument posséder l'entête donné ci-dessus. Dans le cas contraire, votre classe ne pourra pas être testée correctement, et vous risquez de perdre tous les points accordés à l'exécution.

2.1.2 ATTRIBUTS D'INSTANCE

Votre classe `FilePrioChainee` doit posséder deux attributs d'instance nommés `elements` et `taille` :

Nom attribut	Type	Description
`elements`	`Maillon<T>`	Le premier maillon de la liste chainée représentant la tête de la file de priorité. Si `elements` est `null`, c'est que la file de priorité est vide.
`taille`	`int`	Le nombre d'éléments dans cette file de priorité. Il doit être mis à jour à chaque ajout ou retrait d'éléments dans la liste chainée représentant la file de priorité. Lorsque `taille` contient la valeur 0, c'est que la file de priorité est vide.

***Notes** :*

- *Aucun autre attribut d'instance n'est permis.*
- *Il est important de respecter le nom, et le type de ces attributs, sinon les tests ne compileront pas et vous perdrez beaucoup de points.*

2.1.3 CONSTRUCTEUR

Votre classe doit posséder un seul constructeur, sans argument, qui crée une file de priorité vide (qui ne contient aucun élément). Ce peut être le constructeur par défaut.

2.1.4 MÉTHODES D'INSTANCE PUBLIQUES

Comme la classe `FilePrioChainee` implémente l'interface `IFilePrio` (fournie avec l'énoncé du TP), elle doit donc implémenter toutes les méthodes listées dans cette interface.

***Notes** :*

- *Lisez bien la Javadoc de chaque méthode dans l'interface `IFilePrio` pour bien comprendre et respecter ce qu'elles doivent faire.*
- *Les méthodes à implémenter qui retournent une `IFilePrio`, doivent retourner une instance de `FilePrioChainee`.*

En plus des méthodes de l'interface `IFilePrio`, vous devez ajouter dans votre classe `FilePrioChainee` une redéfinition de la méthode `toString`. **Celle-ci vous est fournie dans un fichier texte accompagnant l'énoncé du TP. Vous DEVEZ simplement la copier-coller dans votre classe `FilePrioChainee`**.

***Note** : Cette méthode `toString` sera possiblement utilisée telle quelle dans les tests de votre classe, donc elle doit être présente et vous ne devez pas la modifier. Elle vous sera aussi utile pour vos propres tests.*

Les seules méthodes publiques pouvant se trouver dans la classe `FilePrioChainee` sont celles listées dans l'interface `IFilePrio` en plus de la méthode `toString`. **Toute autre méthode doit être privée.**

2.2 Tests de votre classe `FilePrioChainee`

Vous n'avez pas d'application à remettre, cependant, vous devrez coder des programmes / méthodes pour tester votre classe `FilePrioChainee`. Comme les éléments contenus dans la file de priorité doivent implémenter l'interface `ITachePrio`, vous pouvez utiliser la classe `TachePrio` (fournie avec l'énoncé du TP) qui implémente cette interface. Vous pourrez donc instancier une file de priorité contenant ce type d'éléments.

3. Interfaces, classes, et méthode fournies (**à ne pas modifier**)

1) Interface `IFilePrio`
2) Interface `ItachePrio`
3) Classe `Maillon`
4) Classe `TachePrio`
5) Classe `FileVideException`
6) Méthode `toString` à copier dans votre classe `FilePrioChainee`.

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

4. Précisions supplémentaires pour `FilePrioChainee`

- Vous devez respecter le **principe d'encapsulation** des données.
- N'oubliez pas d'écrire les commentaires de documentation (**Javadoc**) pour documenter vos méthodes et votre classe.
- **À part les attributs d'instance, aucune variable de classe n'est permise**. Vous pouvez cependant ajouter des constantes de classe si vous le jugez pertinent.
- Réutilisez votre code autant que possible. Vous pouvez (et devriez) faire des **méthodes privées** pour bien **structurer/découper votre code** (séparation fonctionnelle).
- Votre code doit compiler et s'exécuter avec le **JDK 8.** Si vous utilisez ce qu'on voit au cours, ce sera le cas.
- Votre classe doit se trouver dans le **paquetage par défaut**.
- Vous DEVEZ respecter le style Java.

- **N'oubliez pas d'écrire (entre autres) votre nom complet et votre code permanent dans l'entête de vos classes.**

Le non-respect de toute spécification ou consigne se trouvant dans l'énoncé de ce 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 **CriteresGeneraux DeCorrection_v2.pdf**. De plus, votre code sera noté sur le respect des conventions de style Java (dont un résumé se trouve dans le document **ConventionsStyleJavaPourINF1120.pdf**). Ces deux documents peuvent être téléchargés dans la section TRAVAUX PRATIQUES (ET BOITES DE REMISE) de 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 la **qualité de votre code** (**incluant la documentation - Javadoc**), et 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 possibles.

6. Date et modalités de remise

6.1 Remise du travail

Date de remise : Au plus tard le **20 mars 2024** à 23h59
Le fichier à remettre : `FilePrioChainee.java`

Remise via Moodle uniquement.

Vous devez remettre (téléverser) vos fichiers sur le site du cours (Moodle). Vous trouverez la boite de remise dans la section **TRAVAUX PRATIQUES (ET BOITES DE REMISE) – BOITE DE REMISE DU TP2**.

NE PAS ARCHIVER le fichier à remettre (PAS DE .zip, .rar, etc.).

Vérifiez deux fois plutôt qu'une **que vous avez remis le bon fichier**, car c'est celui-là uniquement qui sera considéré pour la correction.

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 des classes 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 !