Travail pratique #1

Ce travail doit être fait individuellement.

Notions mises en pratique : implémentation d'une interface (définissant une TDA), généricité, utilisation de la classe `ArrayList`.

1. Description générale

1.1 DÉFINITION DU TDA FILE DE PRIORITE

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)`.

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 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.

Mélanie Lord
lord.melanie@uqam.ca

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`
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 (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.

1.2 UTILISATION DE CETTE FILE DE PRIORITE

Bien que vous n'ayez pas à implémenter un programme qui utilise une file de priorité comme structure de données, voici tout de même un exemple vous montrant une utilisation possible. Imaginez un programme de triage des patients dans une salle d'urgence qui doit permettre de traiter les patients selon leur ordre d'arrivée ainsi que la gravité de leur cas. À mesure que les patients se présentent à l'urgence, on détermine leur priorité de traitement selon la gravité de leur cas. Votre programme pourrait alors utiliser une telle file de priorité pour "enfiler" les patients selon leur priorité assignée (et leur ordre d'arrivée). Les patients les plus prioritaires se trouvent toujours en tête de file. On peut alors défiler les patients l'un après l'autre pour les traiter. Imaginons maintenant que, par manque de médecins spécialistes ou manque de salles opératoires, on ne puisse pas traiter tout de suite le prochain patient le plus prioritaire dans la file, on passerait alors au patient le plus prioritaire de la priorité suivante, c'est-à-dire celui qui se trouve en tête de la file interne de la seconde priorité la plus grande (on "défilerait" le patient de cette file interne), et ainsi de suite.

2. Implémentation et tests

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 **`FilePrio`**, en utilisant la classe `ArrayList` comme structure de données sous-jacente. 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 `ArrayList` dans laquelle la position `0` est la tête de la file, et la position `(size() - 1)` est la fin de la liste. Vous devez donc utiliser une liste `ArrayList<T>` où `T` est le type générique des éléments contenus dans la file de priorité. De plus, ce type générique doit être un type qui implémente l'interface `ITachePrio` fournie avec l'énoncé du 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 être écrit comme ceci :

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

2.1.2 Attribut d'instance

Votre classe `FilePrio` doit ne posséder qu'un seul attribut d'instance nommé `elements` :

Nom attribut	Type	Description
`elements`	`ArrayList<T>`	Les éléments de cette file de priorité représentés en mémoire par une `ArrayList`. La tête de file est à la position 0 de cette `ArrayList` et la fin de la file se trouve à la position `size() - 1` de cette `ArrayList`.

Notes :

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

2.1.4 Méthodes d'instance publiques

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

Note *: Lisez bien la Javadoc de chaque méthode dans l'interface* `IFilePrio` *pour bien comprendre et respecter ce qu'elles doivent faire.*

En plus des méthodes de l'interface `IFilePrio`, vous devez ajouter dans votre classe `FilePrio` 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 `FilePrio`.

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.*

Les seules méthodes publiques pouvant se trouver dans la classe `FilePrio` 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 FILEPRIO

Vous n'avez pas d'application à remettre, cependant, vous devrez coder des programmes / méthodes pour tester votre classe `FilePrio`. 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.

Exemples :

```
//Instanciation de quelques éléments de différentes priorités
TachePrio e1 = new TachePrio(3, "e1");
TachePrio e2 = new TachePrio(7, "e2");

//Instanciation d'une file de priorité
IFilePrio<TachePrio> fp = new FilePrio<>();
//ou FilePrio<TachePrio> fp = new FilePrio<>();

//Utilisation de la file de priorité (tests)
fp.enfiler(e1);
fp.enfiler(e2);

//affichage de la file pour voir si l'enfilement se fait correctement
System.out.println(fp);

//etc.
```

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

1) Interface `IFilePrio`
2) Interface `ITachePrio`
3) Classe `TachePrio`
4) Classe `FileVideException`
5) Méthode `toString` à copier dans votre classe `FilePrio`.

Les fichiers fournis se trouvent dans l'archive **fichiersRequis.zip** pouvant être téléchargé sur Moodle, dans la section CONTENU DU COURS / Travaux / TP1.

***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

Vous devez respecter le **principe d'encapsulation** des données.

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 faire des **méthodes privées** pour bien structurer/modulariser le code (séparation fonctionnelle).

Votre code doit compiler et s'exécuter avec le **JDK 7**.

Il ne doit y avoir aucun affichage dans vos méthodes (pas de `System.out`)

Votre classe doit se trouver dans le **paquetage par défaut**.

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**" dans la section CONTENU DU COURS / Travaux, sur Moodle.

De plus, votre code sera noté sur le respect du style Java, dont un résumé se trouve aussi dans la section CONTENU DU COURS / Travaux, sur Moodle.

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.

Note : *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.*

6. Date et modalités de remise

6.1 REMISE

Date de remise : **25 février 2017** avant minuit (ou le 26 février À MINUIT)
Le fichier à remettre : `FilePrio.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 **BOÎTES DE REMISE - TRAVAUX PRATIQUES / REMISE DU TP1**

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

- **N'oubliez pas d'écrire (entre autres) votre nom complet et votre code permanent dans l'entête de la classe à remettre.**
- Aucun programme reçu par courriel ne sera accepté (à moins d'une entente préalable).
- Vous avez la responsabilité de conserver des copies de sauvegarde de votre travail (sur disque externe, Dropbox, Google drive, etc.). La perte d'un travail due à un vol, un accident, un bris… n'est pas une raison valable pour obtenir du temps supplémentaire pour la remise de votre travail.
- 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 !