IUP GMI 2 – 2005/2006

Devoir Surveillé de CLPO

Le 13 mars 2006 – durée 1h30

---

il n'y a qu'un seul exercice. Lisez entièrement l'énoncé avant de commencer à répondre aux questions.

Exercice 1 : Une entreprise dispose d'un service informatique, qui développe des applications pour toute l'entreprise. Ce service informatique a besoin de connaître précisément le coût de chacun des projets qu'elle réalise. Pour cela, elle calcule le coût journalier de chaque collaborateur travaillant pour elle. Le service informatique utilise les talents de différents types de collaborateurs :

- Des salariés de l'entreprise. Pour un salarié, on connaît son salaire brut, ce n'est donc pas difficile d'en déduire un coût journalier, sachant qu'il y a 12 mois dans une année, soit 365 jours.
- Des informaticiens extérieurs à l'entreprise, en "régie" dans le service informatique. Ces informaticiens sont facturés à la journée. On connaît donc précisément leur coût journalier.
- Des consultants, qui sont également payés à la journée mais qui font aussi payer un coût forfaitaire pour chaque projet (indépendant du nombre de jours). Le coût d'un projet de n jours est donc n fois le coût journalier plus le coût forfaitaire.

L'interface ci-dessous décrit des méthodes que l'on souhaite présentes pour tout type de collaborateur :

```
interface Collaborateur {
  double coutProjet(int n) ;
  double coutJournalier() ;
  String nom() ;
}
```

La méthode `coutProjet` renvoie le coût du collaborateur employé pour un projet de n journées. La méthode `coutJournalier` renvoie le coût journalier du collaborateur. Pour un consultant, ce tarif ne tient pas compte du coût forfaitaire.

Question 1.1 : Ecrire les classes `Salarie`, `EnRegie`, `Consultant`, qui représentent ces trois types de collaborateurs. Toutes ces classes implémentent l'interface `Collaborateur`. Vous pouvez bien sûr écrire d'autres classes si nécessaire (par exemple pour factoriser du code).

Question 1.2 : On veut pouvoir trier les collaborateurs selon leur nom. Définissez un *ordre naturel* sur les collaborateurs, qui soit l'ordre lexicographique de leur nom (donc, l'ordre naturel sur la classe `String`). Dites précisément quel code java vous devez ajouter dans les classes décrivant les collaborateurs.

On veut maintenant représenter le service informatique par une classe `ServiceInfo` qui dispose d'une variable d'instance tableau de collaborateurs (surdimensionné, par exemple de taille 500).

Question 1.3 : Donner le code de la classe `ServiceInfo`, qui disposera des méthodes suivantes :

- La méthode `void ajouter(Collaborateur c)`
  ajoute un collaborateur s'il n'est pas déjà présent dans le tableau, et ne fait rien s'il est présent. On laissera une exception `ArrayIndexOutOfBoundsException` se déclencher si le tableau est plein.
- La méthode `void retirer(Collaborateur c)`
  retire un collaborateur du tableau s'il est présent, sinon ne fait rien.
- La méthode `int nbCollaborateurs()`
  renvoie le nombre de collaborateurs du service.
- La méthode `Collaborateur[] tous()`
  renvoie un tableau contenant tous les collaborateurs, triés par ordre lexicographique de leur nom. Ce tableau a pour taille `nbCollaborateurs()`.

Avant le démarrage d'un projet de n jours, le service informatique aimerait connaître les collaborateurs qui coûtent le moins cher.

Question 1.4 : Définir une classe implémentant `java.util.Comparator` et permettant de comparer deux collaborateurs selon leur coût pour un projet de n jours.

Question 1.5 : Ajoutez à la classe `ServiceInfo` une méthode
`Collaborateur[] tousParCout(int n)`
Cette méthode renvoie un tableau contenant tous les collaborateurs, triés par ordre croissant de coût pour un projet de n jours. Ce tableau a pour taille `nbCollaborateurs()`.

Annexe

L'interface Comparable

Cette interface ne comporte qu'une seule méthode :

```
public int compareTo(Object o)
```

L'interface java.util.Comparator

```
public int compare(Object o1, Object o2);
public boolean equals(Object o)
```

Rappelons que la méthode `equals` sert à comparer le comparateur à un autre (donc renvoie vrai si et seulement si `o` est un comparateur qui définit la même relation d'ordre).

Quelques méthodes de java.util.Arrays

Cette classe contient des méthodes qui permettent de manipuler des tableaux. En particulier, on y trouve des méthodes de tri :

```
static void sort(Object[] a)
          Trie le tableau a par ordre croissant,
          selon l'ordre naturel de ses éléments.

static void sort(Object[] a, Comparator c)
          Trie le tableau a par ordre croissant,
          selon l'ordre défini par le comparateur c.
```