Choix de TDA : exercices

Pour toutes les situations suivantes, on vous demande de compléter les classes Java proposées afin de garantir une efficacité maximale des différentes méthodes demandées.

Donnez le **coût** de chaque méthode demandée.

Chaque exercice propose une classe de test qui vérifie sommairement que les fonctionnalités demandées sont correctes.

1. Ordonnancement des tâches

Le chef du service a comme rôle de répartir différentes tâches à ses employés. Il y a des tâches qui sont plus urgentes que d'autres. Quand un employé termine une tâche, il se présente chez son chef qui lui attribue une nouvelle tâche.

En vue de l'informatisation, voilà ce qui a été décidé :

- Le chef de service encode les tâches. Pour chaque tâche, il introduit un descriptif et une priorité.
- Le descriptif est une chaîne de caractères.
- La priorité est un chiffre compris entre 1 et 5. La priorité 5 est la plus importante.
- Quand deux tâches ont la même priorité, c'est la plus ancienne (celle qui a été encodée en premier) qui sera d'abord exécutée.
- Quand un employé termine une tâche, il se voit automatiquement (le programme) attribué une nouvelle tâche. Il s'agit donc de la tâche la plus ancienne de la priorité la plus grande.

Complétez la classe Ordonnancement.java

2. Système de contrôle d'accès

Une société désire mettre en place un système de contrôle d'accès de ses employés à l'aide de badges qu'ils devraient scanner lors de leurs entrées et sorties du bâtiment.

Cette société est essentiellement intéressée par savoir qui est présent dans le bâtiment.

Complétez la classe ControleDAcces.java

3. Location de Patins

Lors des fêtes de fin d'année, une patinoire est installée. Des patins à glace sont mis à la disposition des patineurs. Il y en a de la pointure 33 à 48. Les organisateurs voudraient automatiser la location de ces patins.

Pour ce faire, des casiers seront placés. Ils seront numérotés à partir de 0. Chaque casier contiendra une paire de patins. Une personne désireuse de louer des patins ira à une borne informatisée et recevra une carte magnétique permettant d'accéder à un casier. Elle y trouvera une paire de patins correspondant à sa pointure. De plus, elle bénéficiera du casier pour y déposer ses chaussures et autres affaires le temps qu'elle patine. A son départ, après avoir vidé le casier et y avoir replacé les

patins, elle retournera à la borne informatisée introduire la carte magnétique et payera la location. Le prix demandé dépendra du temps de la location.

Dans cet exercice, vous pouvez supposer qu'un casier libéré contient bien une paire de patins et que ces patins sont les mêmes (même pointure) que ceux qu'on y avait placés au départ.

On vous demande de compléter la classe LocationPatins.

Le constructeur de la classe recevra en paramètre un tableau précisant la pointure de la paire de patins à glace déposée dans chaque casier. Le rang correspond au numéro du casier.

Par exemple :

43	39	40	40	35	32	46	39

Le casier 0 contient une paire de patins à glace de pointure 43 ; le casier 1 contient une paire de patins à glace de pointure 39 ; …

La méthode `attribuerCasierAvecPatins()` renvoie le numéro d'un casier contenant une paire de patins de la pointure passée en paramètre. Cette méthode renvoie -1 s'il n'y a plus un tel casier.

La méthode `libererCasier()` renvoie le prix de la location du casier dont le numéro est passé en paramètre. Elle utilise la méthode `prix()` fournie.

4. Options (Juin 2017)

En deuxième année, vous devez choisir une option parmi trois possibles. Malheureusement, toutes les options ont un nombre de places limitées. On demande donc aux étudiants de donner leurs préférences parmi les options proposées : ils doivent donner un premier choix, un deuxième choix et un troisième choix. Si la limite d'une option est dépassée, les premiers servis sont les étudiants ayant les meilleures moyennes.

Dans le projet `options`, on vous fournit un squelette de code. La classe `Option` retient le nombre de places de l'option ainsi que tous les étudiants inscrits. Dans `Option`, la méthode `inscrireEtudiant(Etudiant etu)` inscrit l'étudiant à l'option courante si il y a encore de la place. Cette méthode renvoie `false` s'il n'y a plus de place.

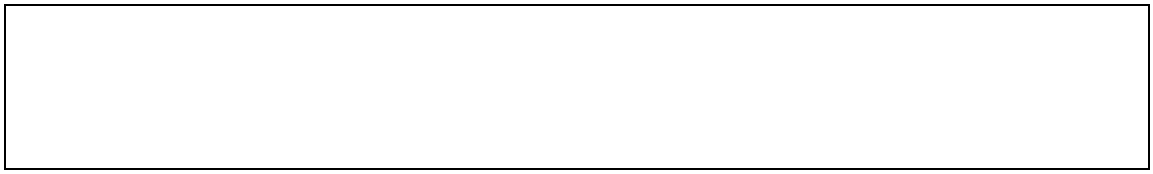
Vous devez compléter la classe `ChoixOptions` en implémentant les méthodes `ajouterPreferences()` et `attribuerOptions()`. `attribuerOptions()` est appelé lorsque toutes les préférences d'options ont été encodées.

Vous aurez besoin de rajouter des attributs dans `ChoixOptions` qui devront être initialisés dans le constructeur. Il n'est pas nécessaire de gérer les cas d'erreur.

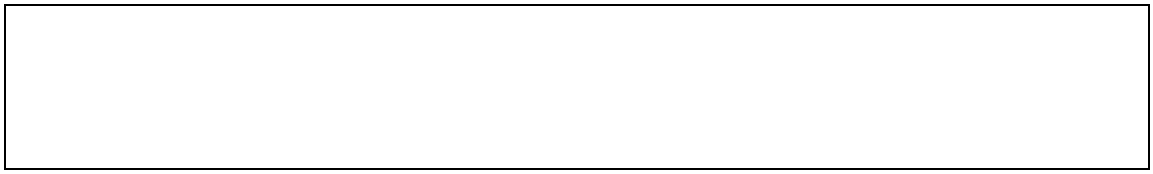
Vous devez garantir une efficacité maximale pour ces deux méthodes.

Dans les cadres ci-dessous, donnez les complexités de vos 2 méthodes :

ajouterPreferences() :



attribuerOptions() :



Une classe `Test` est fournie. L'output attendu est le suivant :

```
SAP
jean
-----------------
Unity
Pas d'inscrit
-----------------
IA
luc paul
-----------------
```

5. Doodle (Aout 2017)

Doodle est un site qui permet de planifier une réunion. Ce site permet de demander les disponibilités de chaque participant à la réunion et de trouver une date qui convient à un maximum de participants.

Dans le projet **`doodle`**, on vous fournit un squelette de code.

La classe `PlageHoraire` représente les plages horaires possibles pour organiser la réunion. Une plage horaire à une heure de début, une heure de fin ainsi que le nombre actuel de participants qui pourraient être présent pour la réunion.

Vous devez compléter la classe Doodle en implémentant les méthodes `ajouterDisponibilites()`, `estDisponible()`, et `trouverPlageQuiConvientLeMieux()`. La méthode `trouverPlageQuiConvientLeMieux()` est appelée lorsque les participants ont donné leurs disponibilités.

Vous aurez besoin de rajouter des attributs dans `Doodle` qui devront être initialisés dans le constructeur. Il n'est pas nécessaire de gérer les cas d'erreur.

Vous devez garantir une efficacité maximale pour ces trois méthodes.

Dans les cadres ci-dessous, donnez les complexités de vos trois méthodes :

`ajouterDisponibilites()`:

`estDisponible()`:

`trouverPlageQuiConvientLeMieux()`:

Une classe `Test` est fournie. L'output attendu est le suivant :

```
Jean est-il disponible pour la première date ? true
Jean est-il disponible pour la deuxième date ? false
Jean est-il disponible pour la troisième date ? false
la plage horaire qui convient le mieux est PlageHoraire [debut=2017-10-
12T14:00, fin=2017-10-12T15:00, nbParticipantPresent=2]
```

6. Gestion des impressions

En section informatique, de nombreux examens se déroulent sur PCs. Pour beaucoup de ceux-ci, l'examen se termine par la remise d'une impression papier. Toutes ces impressions se font sur la même imprimante.

Le temps d'impression peut s'avérer assez long. En effet, certains étudiants lancent plusieurs impressions car ils se rendent compte que l'impression effectuée précédemment contenait des erreurs. Le temps d'impression pourrait déjà être réduit en évitant les impressions inutiles.

On vous demande de mettre en œuvre un nouveau système de gestion des impressions. Ce nouveau système doit bien sûr continuer à respecter l'ordre d'arrivée des impressions. Ce nouveau système doit toujours permettre à un étudiant de lancer plusieurs impressions. Cependant, le système n'imprimera un fichier à condition qu'il n'existe pas d'impression suivante provenant du même compte utilisateur.

Complétez la classe GestionImpression. La méthode `selectionnerImpression()` renvoie la prochaine impression qui doit effectivement être imprimée. Cette méthode renvoie *null*, s'il n'y a plus rien à imprimer.

7. Académie de musique

a) Dans une académie de musique, les élèves peuvent s'inscrire à des cours particuliers d'apprentissage d'un instrument de musique. Évidemment, il n'y a pas toujours de place disponible au moment où l'élève s'inscrit. Dans ce cas, l'élève sera mis en attente. Quand une place se libère pour un cours particulier, c'est l'élève qui s'est inscrit en premier pour cet instrument-là qui aura la place. On vous demande d'écrire l'application qui permet de gérer les files d'attente.

Le constructeur de votre classe recevra un vecteur contenant les noms des différents instruments.

b) Il arrive régulièrement qu'un élève se désiste alors qu'il est en attente pour un cours. Pour cela, il faudrait ajouter à votre classe la méthode :

```
public boolean desistement(String eleve)
```

En ajoutant un TDA supplémentaire, il est possible d'implémenter cette méthode avec un coût en O(1). Les autres méthodes doivent être revues. Leur coût moyen (amorti) ne devrait pas changer !