université PARIS DIDEROT PARIS 7

INITIATION À LA PROGRAMMATION (IP2)

TP5 : Listes simplement chaînées

Dans ce TP, nous étudions les listes simplement chaînées. Dans certains cas, nous prêtons attention à l'ordre des éléments. Lorsque l'implémentation récursive ou itérative de vos méthodes n'est pas imposée par l'énoncé, vous avez le choix.
Nous vous rappelons qu'il est très fortement conseillé de lire tout le TP une fois avant de le commencer. Par ailleurs, vous devez tester régulièrement votre code. Il est normal que vous n'ayez pas toujours le résultat voulu du premier coup, donc n'hésitez pas à faire des essais. Et *surtout*, si vous ne comprenez pas pourquoi un test ne fonctionne pas comme vous vous y attendiez, pensez à utiliser les fonctionnalités de `jGrasp` vues au TP1 pour suivre ce que votre code fait effectivement.

Dans tout ce TP, vous pouvez créer vos propres méthodes supplémentaires si nécessaire, même lorsque cela n'est pas demandé, dans le but de simplifier votre implémentation. Pour simplifier la manipulation des listes, il faut se baser sur le choix effectué en cours : par exemple, une méthode récursive d'affichage de la liste sera implémentée dans la classe de cellule, et initiée à partir de la classe de liste ; ainsi la récursivité est déléguée à la classe de cellule.

Exercice 1 Implémentez :

1. une classe `Employe` avec les attributs suivants :
   - `private final String nom;`
   - `private int salaire;`

   Des accesseurs ("getters") et mutateurs ("setters") pour ces variables lorsque cela est possible, ainsi qu'une méthode d'affichage et un constructeur,
2. une classe `Cellule` avec les attributs
   - `private Employe emp;`
   - `private Cellule suivant;`

   ainsi que deux constructeurs :
   `public Cellule(Employe emp)` et `public Cellule(Employe emp, Cellule suiv)`,
3. une classe `Entreprise` qui contient les attributs
   - `private String nom;` (le nom de l'entreprise)
   - `private Cellule premier;`

   avec un constructeur et les méthodes suivantes :
   - `public void affiche()`, **récursive**, qui décrit tous les employés,
   - `public boolean appartient(String n)`, **récursive**, teste la présence de l'employé de nom `n` dans la liste,
   - `public void ajout(Employe emp)` qui ajoute un nouvel employé en tête de liste s'il n'est pas déjà présent dans la liste,
   - `public void demission(String n)`, **récursive**, qui retire l'employé de nom `n` dans la liste. On suppose qu'il n'existe pas deux employés ayant le même nom dans la liste.

Exercice 2 [Des méthodes utilisant les salaires]

1. Créez une méthode **récursive** `public boolean augmente(String nom, int montant)` qui augmente l'employé `nom`, s'il existe, d'un `montant` strictement positif. La méthode renvoie `false` si l'une des conditions n'est pas respectée.
2. Créez une méthode `public ArrayList<Employe> choixSalaire(int min, int max)` qui renvoie un ArrayList des Employes dont le salaire est compris entre `min` et `max`. N'hésitez pas à consulter la documentation de la classe `ArrayList` si nécessaire.
3. Testez toutes vos méthodes dans un `main`.

Exercice 3 [Liste d'employés ordonnée par salaire] Nous considérons maintenant des `Employes` classés en ordre croissant de leur salaire.

1. Modifier les méthodes implémentées précédemment pour tenir compte de ce nouveau critère : chaque opération doit maintenir l'ordre de la liste.
   (a) `ajout(Employe emp)` doit placer l'employé `emp` après un employé qui gagne moins et avant un employé qui gagne plus, ou en début ou fin de liste, selon les cas, si de tels employés n'existent pas. On demande ici à ce que cette méthode soit **récursive**.
   (b) (Facultatif) `augmente(String nom, int montant)` doit faire remonter l'employé `nom` dans la liste à l'emplacement adéquat (pour préserver l'ordre croissant des salaires).
2. (Facultatif) Ajoutez la méthode `acquisition(Entreprise ent)`, qui ajoute les employés de l'entreprise `ent` à l'entreprise courante, tout en préservant l'ordre croissant. On suppose que chaque employé n'est présent que dans une seule entreprise à la fois.
   Pouvez-vous rendre votre opération plus efficace, notamment en utilisant l'ordre croissant ? Si oui, comment faire ?
   Remarque : cette méthode consiste à fusionner deux listes croissantes de sorte à obtenir une nouvelle liste croissante, sans redondance (chaque élément est présent une unique fois dans la liste obtenue).
3. Y a t-il d'autres méthodes que vous pouvez modifier pour tenir compte de ce nouveau critère, que vous pouvez simplifier, ou rendre plus rapides ?
4. Créer une méthode **récursive** `public boolean croissante()` qui renvoie `true` si et seulement si les salaires sont en ordre croissant dans la liste.
5. Testez toutes vos méthodes dans un `main`.
6. (Facultatif) Créer la méthode `public static Entreprise trierParSalaires(Entreprise e)` qui renvoie une nouvelle instance d'`Entreprise` dont les `Employes` sont ceux de `e` triés par ordre croissant des salaires.