université PARIS DIDEROT PARIS 7

INITIATION À LA PROGRAMMATION (IP2)

Séance 2 : Récursion sur les listes simplement chaînées

Pour cette séance nous vous demandons encore d'écrire des **solutions récursives**.
Il est normal que vous n'ayez pas toujours le résultat voulu du premier coup, continuez à faire des essais, des schémas. Après plusieurs relectures et ajustements vous arriverez à une formulation dont vous serez satisfait.
On rappelle que bien souvent il est nécessaire d'initier le raisonnement récursif dans la classe `Liste`, et de le mettre en place soigneusement dans une nouvelle méthode, écrite dans la classe `Cellule`. (Vous avez vu cela plusieurs fois déjà)

Exercice 1 Implémentez :

1. une classe `Employe` avec les attributs suivants :
   — `private final String nom;`
   — `private int salaire;`
   ajoutez un constructeur, des accesseurs ("getters") et mutateurs ("setters") lorsque cela est possible, complétez par une méthode d'affichage
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 employé en tête de liste s'il n'y a en a pas déjà un de même nom dans la liste,
   — `public void demission(String n)`, **récursive**, qui retire l'employé de nom `n` dans la liste. (et on suppose encore qu'il n'existe pas deux employés ayant le même nom dans la liste)

Exercice 2 [Des méthodes utilisant les salaires]

1. Écrivez 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. **(**)** Réfléchissez à une méthode `public Entreprise choixSalaire(int min, int max)` qui renvoie la partie de l'entreprise constituée des Employes dont le salaire est compris entre `min` et `max`, au sens large.

Exercice 3 [Liste d'employés ordonnée par salaire]

Les méthodes générales de tri ne sont pas au programme de ce semestre, mais nous pouvons supposer dans la suite de cet exercice que les `Employes` sont effectivement déjà classés en ordre croissant de leurs salaires, et nous allons préserver cet ordre dans nos opérations.

1. Écrivez une méthode **récursive** `public boolean croissante()` qui renvoie `true` si et seulement si les salaires sont en ordre croissant dans la liste.
2. Reprenez le code de `ajout(Employe emp)`, qui doit maintenant placer l'employé `emp` après un employé qui gagne moins ou autant, et avant un employé qui gagne plus.

   Si de tels employés n'existent pas, placez le en début ou fin de liste, selon les cas.

   (Pour simplifier et se concentrer sur le problème de l'ordre, on ne testera pas le fait qu'un employé de même nom est ou n'est pas déjà présent)
3. Ecrivez une méthode `acquisition_Version_1(Entreprise ent)`, qui ajoute les employés de l'entreprise `ent` à l'entreprise courante, tout en préservant l'ordre croissant. Dans cette première version vous utiliserez la méthode `ajout` précédemment écrite.

   L'entreprise donnée en argument restera inchangée.

   Pour simplifier, on suppose toujours que dans les données fournies chaque employé n'est présent que dans une seule entreprise à la fois.
4. Pour préserver l'ordre, il nous faut également reprendre l'écriture de la méthode `augmente(String nom, int montant)` (montant est supposé toujours positif).
   Écrivez une première version `augmente_Version_1(String nom, int montant)` qui combine une modification de `demission` et de `ajout`.
5. **(**)** Pour éviter de parcourir deux fois la liste, et à titre d'exercice, nous allons écrire une autre solution `augmente_Version_2(String nom, int montant)`.
   Elle procédera en deux étapes :
   (a) la recherche de la cellule correspondant à l'employé
   (b) la poursuite du traitement en appelant, sur cette cellule, une méthode récursive `pushIt()` qui permutera l'employé avec le suivant tant que c'est nécessaire.
6. **(***)** Ecrivez une nouvelle méthode `acquisition_Version_2(Entreprise ent)` qui corrige un défaut d'efficacité. En effet, l'ajout fait dans la première version recherche à chaque fois une place pour un nouvel employé en oubliant qu'ils étaient déjà ordonnés. Reprenez tout pour que cette acquisition se fasse plus intelligemment.