Pensez à tester vos fonctions au fur et à mesure dans une classe séparée

Exercice 1 Le but de ce TP est d'implémenter une `Entreprise` vue comme une liste chaînée d'`Employes`. Pour cela, on considère la classe `Employe` suivante :

```
class Employe
{
  private String nom;              // le nom de l'employé
  private int salaire;             // salaire de l'employé
  private int benefice;            // bénéfice apporté à l'entreprise

  public Employe()
  {
    this.nom="pas de nom";
    this.salaire=2000;
    this.benefice=0;
  }
}
```

1. Écrire le constructeur `public Employe(String nom, int salaire, int benefice)` construisant un employé avec les valeurs passées en paramètre.

2. Écrire une méthode `public void affiche()` affichant les attributs d'un `Employe` sous la forme : `L'employe <nom> gagne un salaire de <salaire> euros mensuels et apporte un benefice de <benefice> a l'entreprise` .

3. Rajoutez les getters et setters associés à `nom`, `salaire` et `benefice`. Il ne devra pas être possible d'attribuer un salaire inférieur à 1500.

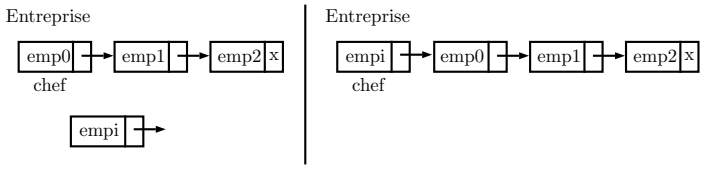
On considère à présent les deux classes suivantes :

```
class Cellule
{
  Employe emp;
  Cellule suivante;
}

class Entreprise
{
  Cellule chef;

  public Entreprise ()
  {
    this.chef = null ;
  }
}
```

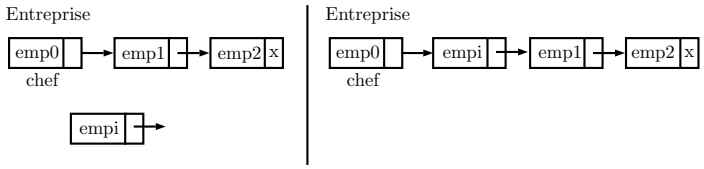
4. Écrivez un constructeur de cellule prenant en argument un `Employe`. Comment allez-vous initialiser l'attribut `suivante` ?

5. Ajoutez à la classe `Entreprise` un constructeur `public Entreprise(Employe e)` créant une entreprise dont l'`Employe` de la `Cellule chef` sera `e`.

6. Écrivez une méthode `public void changerChef(Employe empi)` qui remplace le `chef` de `this` par `empi` et fait de l'ancien chef un employé normal.

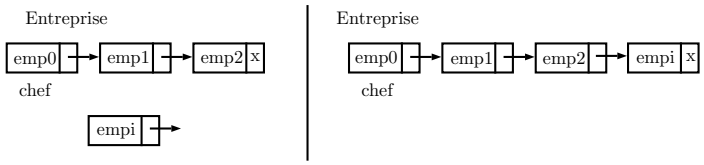


7. Créez une méthode `public void ajout(Employe empi)` qui ajoute un `Employe` à l'`Entreprise`. Cet `Employe` sera ajouté en tête de la liste chaînée, juste après le `chef`.



8. Écrivez une méthode `public void affiche(Entreprise ent)` qui parcourt l'`Entreprise` depuis son chef jusqu'à la fin de la liste et affiche la description de chaque employé sur un ligne (dans le même format que celui de la question 2).

9. Créez une méthode `public void recrute(Employe empi)` qui ajoute un `Employe` à l'`Entreprise` en fin de liste chaînée.



10. Créez une méthode `public void Augmentation()` qui augmente de 10% les salaires de tous les `Employes` de l'`Entreprise`.

11. Écrivez une méthode `public boolean recherche(String nom)` retournant un booléen indiquant si il existe un employé s'appelant `nom` dans l'`Entreprise`.

12. Écrivez une méthode `public void demission(String nom)` correspondant à la suppression de l'`Employe` s'appelant `nom` (s'il est dans l'entreprise).

12. Écrivez une méthode `public Employe iEme(int i)` retournant l'`Employe` à la ième position après le chef s'il existe. Dans le cas contraire retournez un `Employe` construit avec le constructeur sans arguments.

13. Écrivez une méthode `public void acquisition(Entreprise e)` qui ajoute l'`Entreprise e` dans l'`Entreprise` courante. Celle-ci conservera son `chef` et le `chef` de l'`Entreprise` acquise sera intégré comme un `Employe` normal.

14. Écrivez une méthode `public Entreprise startUp()` retournant une `Entreprise` formée de tous les `Employes` dont les salaires sont inférieurs à 2500 et dont les bénefices sont supérieurs à 50.