TD/TP 1 - Employés

Programmation Java : Restrictions d'accès, héritage et polymorphisme.

On va représenter à l'aide de classes en Java des employés d'une entreprise. Chaque employé aura un nom et un genre et peut-être un supérieur hiérarchique. Les cadres sont des employés qui n'ont pas forcément de supérieurs mais éventuellement des subordonnés (au plus 10 par cadre dans un premier temps). Les simples employés (employés qui ne sont pas des cadres) ont obligatoirement un supérieur. On va utiliser pour cela deux classes, la classe **`Employe`** et la sous-classe **`Cadre`** :

```
public class Employe{
    static final int _HOMME = 0;
    static final int _FEMME = 1;

    private int genre;
    private String nom;
    private Cadre superieur;

    public Employe(String nom,int genre, Cadre superieur) {
        ...
    }
    ...
}
```

```
public class Cadre extends Employe {
    private  Employe[] subordonnes = new Employe[10];
    ...
    public Cadre(String nom, int genre) {
        ...
    }
    public Cadre(String nom, int genre, Cadre superieur) {
        ...
    }
    ...
}
```

En TP, au fur et à mesure que vous écrivez les classes, vérifier dans une classe `Principale` contenant la fonction `main` vos différentes méthodes.

Tous les employés doivent avoir les fonctionnalités suivantes :

Q.1) - Des accesseurs publiques aux attributs `genre`, `nom` et `superieur`.
Pour les cadres, le nom retourné par l'accesseur à l'attribut `nom` sera précédé de *Mr* ou *Mme* suivant le genre.

Q.2) - Une méthode `public` **`String`** `toString()` qui calcule une chaîne de caractères pour représenter l'employé :
- pour les cadres, la chaîne commence par la formule de politesse *Mr* ou *Mme* ;
- pour tous elle contient ensuite le nom ;
- pour les employés ayant un supérieur, elle contient ensuite « membre du service de ... ».

Q.3) - Une méthode **`boolean`** `equals(`**`Employe`** `e)` pour tester l'égalité de deux objets **`Employe`**. Ces derniers sont considérés égaux s'ils ont même nom et même genre.

Q.4) - Dans la classe cadre, une méthode **`void`** `ajouteSubordonne(`**`Employe`** `e)` qui permet d'ajouter un nouvel employé au service de ce cadre. Attention, elle doit vérifier qu'il n'est pas déjà présent. Cette méthode doit être utilisée pour compléter le constructeur publique de la classe **`Employe`**.

Q.5) - Dans la classe cadre, une méthode `public void afficheSubordonnes()` qui affiche sur la sortie standard la liste des subordonnés directs de ce cadre.

Q.6) - Une méthode `public double getSalaire()` pour calculer le salaire mensuel brut :
- pour les simples employés, il est égal au salaire de base : 1430,22€ (smic mensuel brut au 1/01/2013) ;
- pour les cadres il est égal au salaire de base plus 40% du salaire de chacun de ses subordonnés directs.

Q.7) - Dans la fonction `main` de la classe **`Principale`** :
- Créer un cadre A qui a pour subordonnés directs un cadre B et deux simples employés.
- Ajouter quatre simples employés au service du cadre B.
- Tester vos méthodes avec ces deux cadres et six simples employés.

Questions facultatives.

Q.8) - Ajouter à la classe **`Employe`** :
- une méthode `public boolean aPourSuperieur(Employe e)` qui teste si l'employé a pour supérieur direct ou indirect l'employé `e`.
- une méthode `public void changementAffectation(Cadre c)` qui change l'affectation de l'employé vers le service du cadre `c`. Ce changement est interdit si l'employé est un supérieur hiérarchique (direct ou indirect) du cadre `c`. Si l'employé était précédemment affecté a un autre cadre, il faut l'enlever des subordonnés de ce cadre.

Q.9) - Ajouter à la classe **`Cadre`** une méthode qui décrit complètement le service de celui-ci sur la sortie standard. La description comprendra la liste complète de tous les subordonnés directs et indirects de ce cadre avec le salaire de chacun.

Q.10) - Enlever la limite de 10 subordonnés directs par cadre. On pourra pour cela utiliser un tableau dynamique de type **`Vector<Employe>`** pour représenter la liste des subordonnés d'un cadre.