Programmation orientée objet (Java)
Exercice Gestion d'entreprise

Exercice 1. Une entreprise souhaite gérer ses employés et ses managers. Tous les employés commencent avec le même salaire minimum fixé à ***7 000 DH*** et tous les managers commencent avec le même salaire minimum fixé à ***10 000 DH***.

Un employé est caractérisé par un nom de type String, une date d'embauche (`dateEmbauche` de type String au format **"yyyy/mm/dd")** et un salaire de type double.

Un manager est un employé qui dirige un secteur. Il est caractérisé par une date d'entrée en fonctions désignant la date à laquelle il a été nommé manager du secteur (`dateFonction` de type String au format "`yyyy/mm/dd`").

On souhaite pouvoir évaluer
- l'ancienneté d'un employé avec une méthode `public int anciennete(String dateCourante)` qui retourne le nombre de mois écoulés depuis sa date d'embauche.
- l'ancienneté d'un manager qui correspond au nombre de mois écoulés depuis qu'il a été nommé manager.

Pour cela, on pourra implanter une méthode `protected int nbMoisEcoules(String dateCourante, String dateDebut)` retournant le nombre de mois écoulés entre les deux dates passées en paramètre. On pourra faire appel à la classe Scanner pour décomposer la date comme indiqué en annexe.

On souhaite pouvoir calculer une augmentation de salaire avec une méthode public double `augSalaire (String dateCourante)` qui retourne le montant du salaire actuel augmenté d'une certaine valeur par mois d'ancienneté. Cette valeur de type `int`, que nous appellerons augmentation, devra être la même pour tous les employés. Par défaut, elle sera fixée à **10**. Elle pourra être modifiée avec une méthode `SetAugmentation` à définir sachant que cette modification s'appliquera alors à l'ensemble des employés.
Pour les managers l'augmentation sera fixée par défaut à **15**.

1. Construire les classes `Employe` et `Manager` telles que décrites ci-dessus. On pourra fournir trois constructeurs pour la classe `Employe` : `public Employe()`, `public Employe(String nom,String dateEmbauche)`,`public Employe(String nom,String dateEmbauche, double salaire)`
et un constructeur pour la classe `Manager` : `public Manager(String nom,String dateEmbauche, String dateFonction.`

Compléter les classes de manière à pouvoir fournir l'état d'un employé (son nom, sa date d'embauche et son salaire) sous forme d'une chaîne de caractère ainsi que l'état d'un manager qui a les mêmes caractéristiques auxquelles on rajoute le statut de manager et sa date d'entrée en fonction.

2. Créer une classe `Test` permettant de tester les classes `Employe` et `Manager`. Pour cela on créera un tableau d'éléments de type `Employe` dans lequel on ajoutera deux employés et un manager.
On affichera le contenu du tableau.

Ensuite on procédera une augmentation de salaire à la date d'aujourd'hui puis on affichera de nouveau le contenu du tableau.

Exercice 2. Compléter l'exercice précédent de la manière suivante :

Construire une classe **Entreprise** permettant de gérer un ensemble d'employés.
Les employés pourront être stockés dans un tableau d'objets de type **Employe**.
La classe sera caractérisée par les attributs suivants: **private Employe[]entreprise, private int capacite = 10** (capacité totale du tableau) et **private int nbEmp**(nombre réel d'employés stockés dans le tableau);

La classe fournira un constructeur sans paramètre ainsi que les méthodes suivantes :
public void ajouterEmp(Employe e) permettant d'ajouter un employé à l'ensemble de employés. On pourra prévoir une méthode **private void redimensionner ()** qui doublera la capacité du tableau si nécessaire.

- **public Employe rechercherEmp(String nom)** permettant de rechercher un employé à partir de son nom passé en paramètre. Vous implémenterez la méthode **public String getNom()** dans la classe **Employe**.

- **public void supprimerEmp(Employe e)** permettant de supprimer un employé de l'ensemble des employés.
  On suppose que l'ordre de stockage des employés est indifférent. Supprimer un employé se fera en écrasant l'employé à supprimer par le dernier employé du tableau.
  Quelle méthode doit-on redéfinir dans la classe **Employe** ? Redefinissez-là. On suppose que deux employés sont égaux s'ils ont même nom, même date d'embauche et même salaire.

- **public boolean estVide()** retournant vrai si le tableau ne contient aucun employé et faux sinon.
- **public void augmenterSalaires(String dateCourante)** qui consiste à appliquer une augmentation de salaire à tous les employés à la même date.

- **public String toString()** qui décrit l'ensemble des employés sous forme d'une chaîne de caractères, un employé par ligne.

Exercice 3. On souhaite gérer les salaires d'un ensemble d'employés dont les données sont stockées dans unfichier **Employe.txt**. Chaque ligne de ce fichier est de la forme **nom dateEmbauche salaire**.
Exemple de ligne du fichier : **Ben Ali ; 2009/09/08 ; 7550**
Construire une classe **GestionSalaires** fournissant les deux méthodes suivantes :

- **public static Entreprise lireDonnees(String nomFichier)** qui retourne un ensemble d'employés de type Entreprise construit par lecture de chaque ligne du fichier passé en paramètre.
  Quelle exception doit lever cette méthode?
  Afin de créer un objet de type Employe à partir de la ligne lue, on ajoutera une méthode lireEmploye dans la classe **Employe** : **public void lireEmploye(String entree)**.
  On pourra utiliser la classe **Scanner** pour lire les lignes à partir du fichier et lire les données d'un employé à partir de la chaîne de caractères entree (voir l'annexe).

- **public static void main(String[] args)** qui permet de lire le fichier Employe.txt. A partir du l'objet de type Entreprise obtenu, afficher les informations de tous les employés puis effectuer une augmentation de salaire sur tous les employés puis afficher de nouveau les informations de tous les employés.
  Gérer l'exception levée par la méthode **lireDonnees**.

Annexe

La classe **java.util.Scanner**

Depuis la **JDK 5.0**, on a la possibilité de balayer un texte et d'en extraire les éléments de type primitif ou les chaînes de caractères. Un objet de type **Scanner** va décomposer son entrée en mots en utilisant un séparateur, qui par défaut est l'espace vide.

a) Constructeurs utiles de la classe Scanner :

```
Scanner (String source)
Scanner (InputStream source)
```

b) Méthodes utiles de la classe Scanner :

- public boolean hasNext()
- public String next() throws NoSuchElementException
- public boolean hasNextInt()
- public int nextInt() throws NoSuchElementException

On peut lire l'année et le mois d'une date au format "**yyyy/mm/dd**" avec la classe Scanner en utilisant le délimiteur " / " :

```
String date = "2010/11/09";
Scanner s = new Scanner(date).useDelimiter("/");
```

Le premier appel à **s.nextInt()** retournera l'année, le deuxième appel à **s.nextInt()** retournera le mois.

c) Méthodes de la classe Scanner pour lire les entrées

- String s = sc.nextLine();
- String mot = sc.next();
- int n = sc.nextInt();
- double d = sc.nextDouble();

d) La classe Scanner permet de lire les entrées

i) depuis un objet de type String (ex : **String leTexte**)

```
Scanner sc = new Scanner (leTexte);
int i = sc.nextInt();//pour lire l' entier suivant dans leTexte
```

ii) depuis la fenêtre de la console. On construit un objet Scanner attaché à l'unité « d'entrée standard » **System.in** :

```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt(); //pour lire l' entier suivant
String mot = sc.next(); //pour lire le mot suivant
```

iii) depuis un fichier contenant, par exemple, des entiers de type double

```
Scanner sc = new Scanner(new File("DesNombres"));
while (sc.hasNextDouble()) {
      double d = sc.nextDouble();
}
```

La classe Scanner fourni les méthodes permettant d'éviter les erreurs susceptibles de se produire si par exemple, on appelle **sc.nextInt()** alors qu'aucun entier ne peut être lu.

- public boolean hasNextLine()
- public boolean hasNext()
- public boolean hasNextInt()