Secteur Tertiaire Informatique

Filière « Etude et développement »

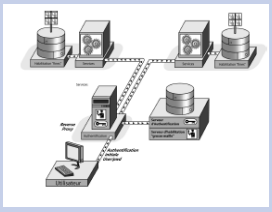
TP – Première application objet

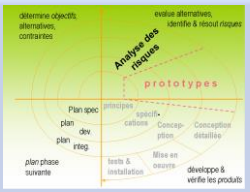
Programmation Orientée Objet

Apprentissage

Mise en situation

Evaluation





1. INTRODUCTION

L'objectif de ce TP est de concevoir un programme en console basé sur une approche objet et permettant de gérer (très basiquement) des salariés d'une entreprise.

Fonctionnalités attendues :

- Pouvoir créer des employés d'une entreprise avec une approche objet ;
- Ajouter des employés à une liste ;
- Afficher certaines informations concernant les salariés.

Le code attendu doit suivre les règles syntaxiques et les conventions de nommage du langage Java.

2. IMPLEMENTATION INITIALE

2.1 IMPLEMENTATION DE LA CLASSE « EMPLOYEE »

Dans un premier temps, il vous faudra créer une classe représentant un salarié d'entreprise.

Cette classe devra être créée dans le package :

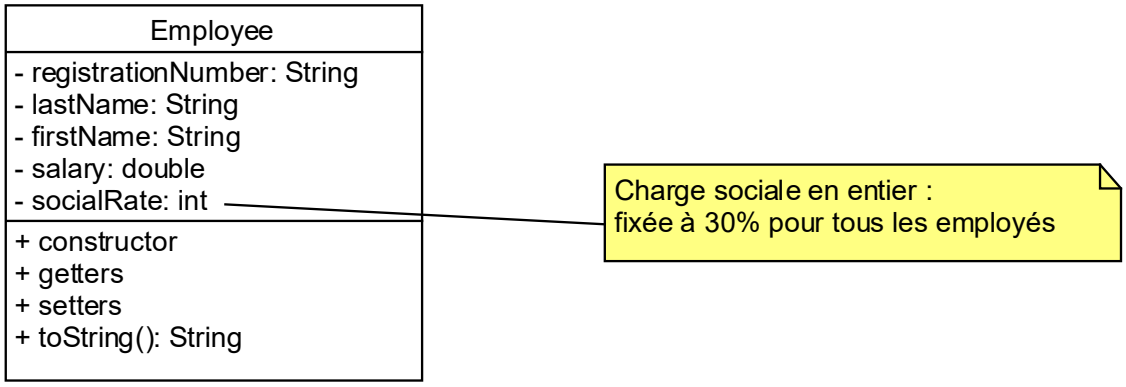
```
package fr.afpa.employees;
```

La classe doit être contenue dans un fichier « .java » portant le même nom.

Pour cette première version, un salarié sera défini par les attributs suivants :

- **Matricule** (« registration number ») : le matricule du salarié est son identifiant unique. La valeur du matricule permet donc de discriminer chaque objet de type salarié.
- **Nom** (« last name »): le nom patronymique du salarié.
- **Prénom (« fist name »)** : son prénom.
- **Salaire brut (« salary »)** : le salaire brut mensuel de référence (sans retrait des charges sociales).
- **Taux de charges sociales (« social participation »)** : le taux de charges sociales (part salariale). Dans un premier temps on considèrera que le taux de charge est de 30%.

Cette première version de la classe employée peut être graphiquement représentée en utilisant le langage UML comme présenté ci-dessous (les noms de variables en anglais pour se rapprocher du code Java) :



A faire

Complétez la classe « Employee » pour y ajouter tous les attributs ainsi que les « **getters** » (appelés accesseurs en français) et les « **setters** » (appelés mutateurs).

Information concernant le taux de charges sociales : dans un premier temps on considèrera que **les charges sont** fixes pour tous les employés et qu'il sera impossible de les modifier.

Vous pouvez ajouter le mot clef « **final** » pour empêcher toute modification d'une variable, ainsi l'attribut pourra être :

```
    private final int socialRate = 30;
```

A faire

Complétez la classe « Employee » pour y ajouter un constructeur prenant en paramètre toutes les valeurs permettant d'initialiser tous les attributs sauf le taux de charges sociales.

A faire

Ajoutez la méthode « **toString()** » qui aura pour objectif de transformer un objet en une représentation sous forme de chaîne de caractères.

Pour plus d'information concernant la méthode « **toString()** » :
https://codegym.cc/fr/groups/posts/fr.986.mthode-java-tostring

2.2 TEST DE LA CLASSE

Une fois la classe implémentée pour pourrez compléter le code de la fonction « **main** » située dans le ficher « EmployeeMain.java ».

A faire

Instanciez 4 objets de la classe « Employee » avec des informations différentes.

Pour chacun des objets, affichez sa représentation en chaîne de caractères dans la console en faisant appel à la méthode « **toString()** »,

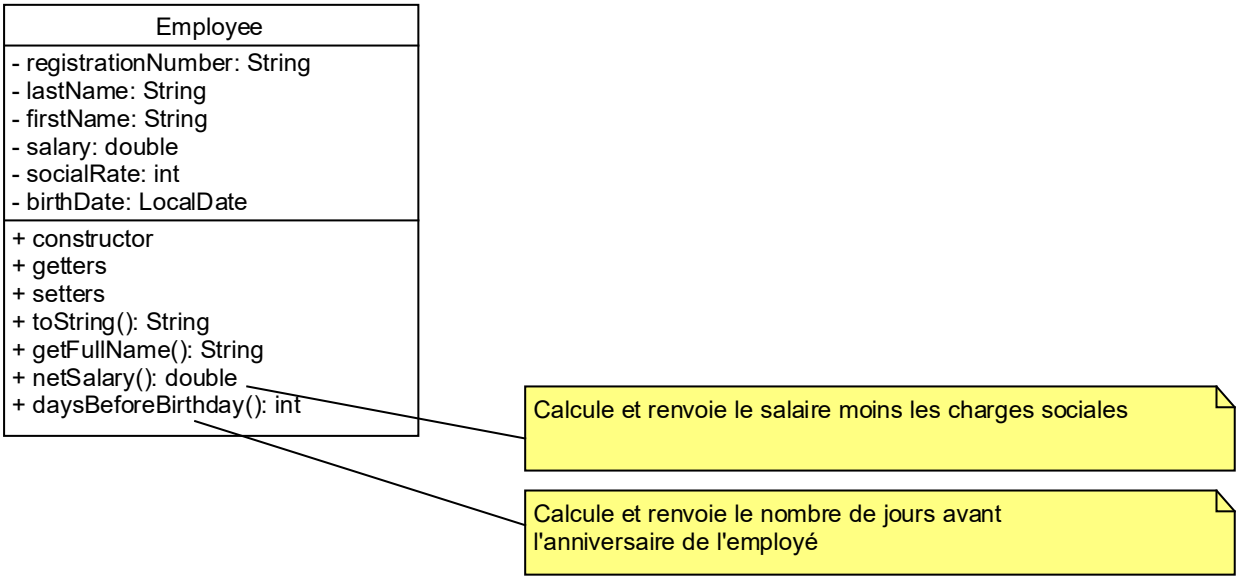
3. AMELIORATION DE LA CLASSE « EMPLOYEE »

3.1 FONCTIONNALITES ATTENDUES

Il vous est demandé d'ajouter des fonctionnalités à la classe précédemment développée :

- Ajout d'une **méthode** permettant d'obtenir le **nom complet** (nom + prénom) d'un employé
- Ajout d'une **méthode** renvoyant le **salaire net** (salaire brut – charges sociales)
- Ajout d'un **attribut** correspondant à la **date de naissance** de l'employé
- Ajout d'une **méthode** renvoyant le **nombre de jours restant** avant la date d'anniversaire de l'employé

Ci-dessous le diagramme UML présentant l'évolution de la classe :



Attention

Pour chacune des fonctionnalités implémentées pensez à tester votre code en faisant évoluer la fonction « **main** ».

3.2 INDICES SUR LE DEVELOPPEMENT

3.2.1 Calcul du salaire net

Dans ce logiciel le calcul du salaire net est loin d'être réaliste.

Vous pourrez utiliser la formule suivante :

Salaire net = salaire brut – salaire brut * taux de charges sociales

3.2.2 Gestion des dates

3.2.2.1 Ajout de l'attribut

Pour le type de l'attribut « **birthDate** » vous pourrez utiliser la classe « **LocalDate** » du package « **java.time** ».

Pour plus d'informations sur la classe « **LocalDate »,** lisez l'article suivant : https://cr10.fr/java-localdate/

Pour exemple, pour déclarer l'attribut vous pourrez utiliser la ligne suivante :

```
private LocalDate birthDate;
```

Question

Les objets de la classe « **LocalDate » sont immuables.**

Qu'est-ce qu'un objet immuable ?

Pour chaque attribut ajouté n'oubliez pas les éléments suivants :

- **« Getter »**
- **« Setter »**
- Le paramètre dans le constructeur
- Ajouter la date à la métode « **toString()** »

Attention

Pour initialiser la date via le constructeur il vous est conseillé de passer en paramètre **une chaîne de caractères** qui contient la date en toutes lettres et d'utiliser une méthode pour la transformer :

```
LocalDate.parse(birthDate);
```

3.2.2.2 Différence entre deux dates

Afin d'implémenter la méthode qui renvoie le nombre de jour restants avant la date anniversaire de l'employé il vous faut deux objets de la classe « **LocalDate** » :

- La date actuelle
- La date d'anniversaire future

Pour obtenir un objet de la classe « LocalDate » vous pourrez utiliser :

```
LocalDate currentDate = LocalDate.now();
```

Pour trouver la différence en jours entre deux dates vous pourrez utiliser la méthode suivante :

```
ChronoUnit.DAYS.between(firstDate, secondDate);
```

Voici un exemple plus complet des solutions pour trouver la différence entre deux dates :

https://mkyong.com/java8/java-8-how-to-calculate-days-between-two-dates/

Attention

Pour chacune des fonctionnalités implémentées pensez à tester votre code en faisant évoluer la fonction « **main** ».

4. VERIFICATION DE LA VALEUR DES ATTRIBUTS

4.1 OBJECTIF DE DEVELOPPEMENT

Vous allez maintenant ajouter des processus de vérification des paramètres lors de la modification ou l'instanciation

Voici ce qu'il faudra vérifier :

- **Le format du matricule** :

  Le matricule est une valeur composée de 7 caractères observant les règles suivantes nnXXXnn où n est un chiffre et X un caractère alphabétique (par exemple, le matricule « 11ABC22 » est correct, « Y5BN22 » est incorrect). Dans un premier temps, cette contrainte ne sera pas prise en compte.

- **Le nom et le prénom** :

  Ne doit pas être vide et ne doit pas contenir de chiffres

- **Le format de la date** :

La vérification de ces données doit pouvoir s'effectuer à chaque tentative de modification de l'attribut concerné :

1. Lors de l'instanciation (dans le **constructeur**)
2. Lors de la tentative de modification de l'attribut (dans les **setters**)

Info

Vous allez implémenter un mécanisme permettant de **vérifier les paramètres des fonctions**.

Ceci est appelé la **PROGRAMMATION DEFENSIVE**

Le premier principe de cette approche est de ne JAMAIS faire confiance aux données entrées par l'utilisateur.

Afin d'adopter une approche de programmation défensive il s'agit de supposer que les données d'entrées auront des valeurs inattendues.

Par exemple :

- Un âge est attendu en paramètre et la fonction reçoit un nombre négatif
- Une fonction utilise un paramètre pour faire une division et 0 lui est envoyé

En résumé : vérifiez tous les paramètres qui peuvent poser problème. Le manque de vérification peut conduire à l'arrêt de votre logiciel.

Pour chacune des données à vérifier vous pourrez créer une méthode de vérification adaptée.

Par exemple, pour le matricule :

```
    private boolean checkRegistrationNumber(String registrationNumber)
```

Cette méthode peut, par exemple, retourner un booléen à « VRAI » si le matricule est bon et « FAUX » sinon.

Si le matricule n'est pas correctement formaté vous pourrez utiliser le système d'exception pour gérer le cas.

A faire

Lisez le cours « Gestion des exceptions » disponible sur Métis pour en apprendre plus sur ce mécanisme et avant de vous lancer dans l'implémentation.

Une exception peut être créée et lancée avec le code suivante (le message est à adapter au besoin).

L'exception interrompt le déroulement de la fonction en cours et le bloc « catch » prend le relais :

```
throw new Exception("Le paramètre d'entrée n'est pas correctement formaté.");
```

Pour déclarer qu'une fonction peut lancer une exception il faut utiliser le mot clef « throws » à ajouter la suite de sa déclaration.

Par exemple, nous pourrons obtenir la déclaration de setter suivante :

```
public void setRegistrationNumber(String registrationNumber) throws Exception
```

Vous retrouverez plus d'informations sur les exceptions au lien suivant : https://devstory.net/10187/java-exception-handling

5. CREATION D'UNE LISTE

Il peut être intéressant de garder une liste des employés.

Pour se faire ajouter l'instanciation d'un objet de la classe « **ArrayList** » et ajoutez-y tous les employés instanciés.

Pour plus d'information sur l'utilisation des « ArrayList » : https://www.w3schools.com/java/java_arraylist.asp

Une fois votre liste créée, vous pourrez afficher successivement les informations de chaque employé en utilisant, par exemple, une boucle « **foreach** ».

Pour un exemple d'utilisation de la boucle « **foreach** » avec une « **ArrayList** » : https://www.data-transitionnumerique.com/foreach-java/

CREDITS

ŒUVRE COLLECTIVE DE l'AFPA
Sous le pilotage de la DIIP et du centre d'ingénierie sectoriel Tertiaire-Services

Equipe de conception (IF, formateur, mediatiseur)
Michel Coulard – Formateur Evry
Chantal Perrachon – IF Neuilly sur Marne

Date de mise à jour : 20/06/2024