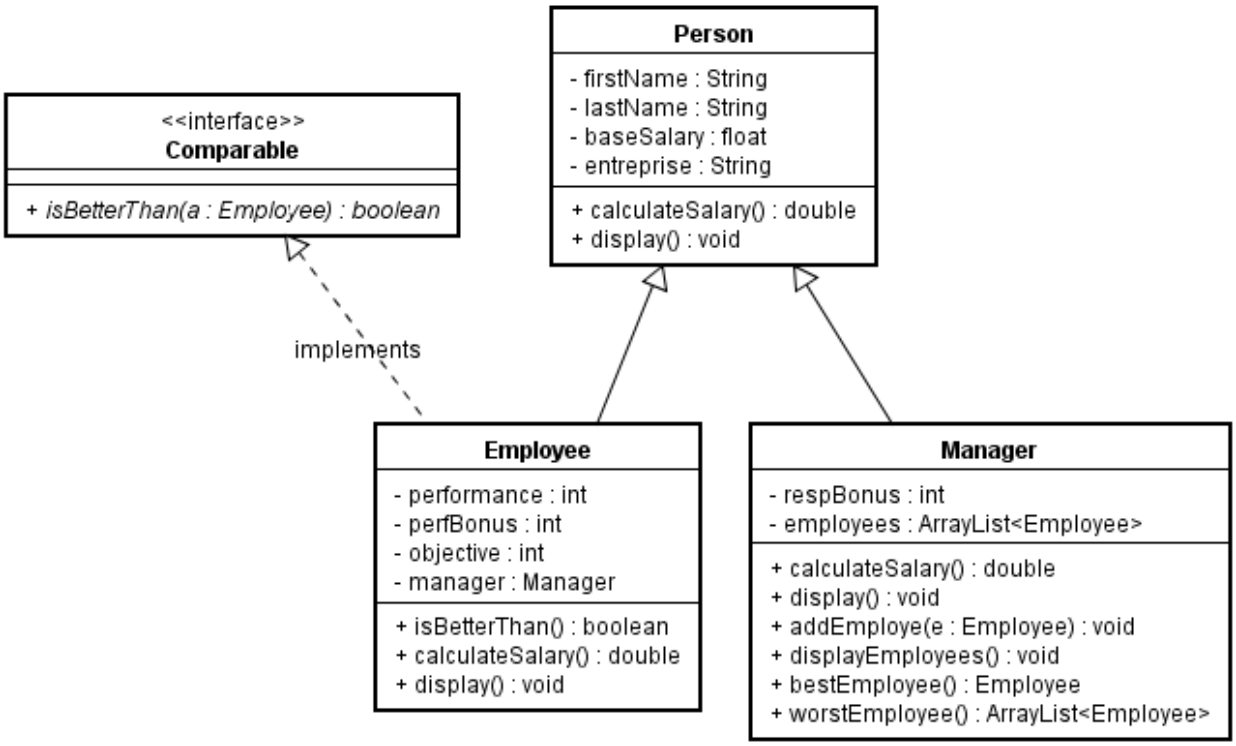
Etude de Cas : Java

Une entreprise « **DataTruc** » souhaite développer une application pour la gestion de ses employés et le suivi de leur performance, elle a donc opté pour une approche orienté objet.

Soit l'extrait de diagramme de classe suivant :



Exercice 1 : Gestion des performances

Spécification fonctionnelle :

Une personne dans cette entreprise est caractérisée par son nom et son prénom et son salaire de base, on doit pouvoir calculer son salaire final et afficher ses informations.

Il existe deux types de personnes dans l'entreprise, les employés normaux et les managers :

- Les employés sont caractérisés par leur indice de performance, ils peuvent bénéficier d'une prime de performance.
- Les managers possèdent une collection d'employés qu'ils gèrent, et bénéficient d'une prime de responsabilité.

Le calcul du salaire global est comme suit :

- Pour les managers il s'agit du salaire de base additionné de la prime de responsabilité
- Pour les employés il s'agit du salaire de base additionné de la prime de performance à condition que leur indice de performance soit supérieur à 1.

L'entreprise désire pouvoir comparer ses employés, un employé est considéré meilleur qu'un autre si sa performance lui est supérieur. Aussi, un manager a la possibilité de savoir qui, parmi les employés qu'il gère, est le meilleur employé ou le plus mauvais employé.

Finalement, la prime de responsabilité du manager doit être au minimum 10% de son salaire, le système ne doit pas permettre la création de manager avec une prime inférieure.

Spécification technique :

Les deux classes **Employee** et **Manager** héritent de la classe abstraite **Personne**. Employee implémente l'interface Comparable.

Chaque **Manager** gère un ensemble d'employés et peut ainsi définir leurs objectifs, évaluer leurs performances etc. La classe **Employee** dispose du constructeur et des setters et getters utiles, ainsi que des méthodes suivantes :

- **void dislay() :** affiche dans la console le nom et le prénom de l'employé
- **int isBetterThan(Employe emp) :** retourne -1 si la performance de l'employé courant est inférieur à la performance de l'employé en paramètre, 0 si leurs performances sont égales et 1 si la performance de l'employé courant est supérieur à la performance de l'employé en paramètre.

La classe **Manager** dispose du constructeur et des setters et getters utiles, ainsi que des méthodes suivantes :

- **void addEmployee(Employee emp) :** ajoute l'employé emp à la liste des employé du manager courant.
- **void displayEmployees():** affiche la liste des employés affectés au manager courant.
- **Employee meilleurEmploye():** retourne l'employé du manager avec la meilleure performance (utiliser la méthode isBetterThan)
- **ArrayList<Employee> worstEmployee():** renvoie la liste des employés du manager dont les performances sont inférieurs à leurs objectifs.

Questions :

- Créer les classes **Person, Employee et Manager** (classe / constructeurs / méthodes / getters / setters)
- Créer une classe **TestEntreprise** qui effectue le traitement suivant :
  - Instancier les objets suivants

Employee					
lastName	**firstName**	**baseSalary**	**perfBonus**	**performance**	**manager**
A	Youssef	11000	3000	300	ahmed
A	Mohammed	10000	3000	200	imane
A	Mouna	11000	3000	100	ahmed

Manager				
lastName	**firstName**	**baseSalary**	**respBonus**	**employees**
B	Ahmed	18000	3000	youssef, mouna
A	Imane	15000	5000	mohammed

  - Afficher les employés du manager Ahmed

- Créer l'interface **Comparable**
- Dans la classe **Employee**
  - Implémenter la méthode isBetterThan
- Créer une classe **ManagerException** qui retourne le message suivant : « Prime incomplète » lorsqu'on essaie de créer un manager avec une prime inférieure à 10% du salaire.
- Modifier la classe **Manager** en conséquence

- **Dans la classe Manager**
  - Implémenter la méthode **bestEmployee**
  - Implémenter la méthode **worstEmployee**
- **Dans la classe TestEntreprise**
  - Afficher le meilleur employé du manager Ahmed
  - Afficher les pires employés du manager Ahmed

- Créer une collection staff qui regroupe tous les collaborateurs de l'entreprise (Managers et Employés)
- Créer une méthode **float payrol(staff)** (payroll est la masse salariale) qui parcourt la collection staff et calcule la masse salariale globale de l'entreprise. On prendra en considération le salaire ainsi que les primes.

Exercice 2 : Gestion des congés

L'entreprise souhaite également automatiser la gestion des demandes et d'acceptation de congé ainsi que des absences de ses collaborateurs.

1. Créer une classe générique **Interim** qui définit :
   a. Un collaborateur qui le remplacera en cas d'absence, l'interim doit être du même type (Employé pour Employé et Manager pour Manager)
   b. Un booléen qui indique la disponibilité du collaborateur
2. Créer dans la classe **Personne** une méthode **boolean demandeAbsence(datedébut, durée, message, interim)** qui pemet de faire une demande d'absence en spécifiant la date de début, la durée de l'absence en jour, un message explicatif libre (exemple : « je souhaite m'absenter pour cause de maladie » ou « je souhaite prendre un congé »), et l'interim. Ce dernier argument peut être **null**.

Pour le traitement des demandes, le programme doit détecter automatiquement le motif de l'absence à partir du message du collaborateur. Ensuite, on applique les règles de gestion suivantes :

- Si le collaborateur indique dans le message de sa demande que le motif de l'absence est une maladie, la demande est acceptée automatiquement sans obligation d'intérim.
- Si le collaborateur indique dans le message de sa demande que le motif de l'absence est un congé, on effectue le traitement suivant :
  - Si la durée est inférieure ou égale à 2 jours, la demande est acceptée sans obligation d'intérim
  - Si la durée est strictement supérieure à 2 jours, la demande est acceptée si l'interim est spécifié et est disponible
- Si le collaborateur n'indique pas de motif dans sa demande, la demande est rejetée

3. Tester la demande

Exercice 3 : Interface graphique et persistance

Dans un SGBD, on suppose que les tables Employee et Manager sont créés comme suit :

Employee				
lastName	**firstName**	**baseSalary**	**perfBonus**	**performance**

1. Créer un formulaire Swing qui permet d'insérer un Employé dans la base de données
2. Ajouter les employés Youssef et Mounia dans la base
3. Dans le formulaire, vérifier que les attributs baseSalary, perfBonus et performance sont bien numériques
4. Créer une fenêtre Swing qui affiche la liste des employés depuis la base de données (utiliser une **JTable**)

---

5. Enregistrer les informations du Manager « Ahmed » (lastName, firstName, baseSalary, respBonus) dans un fichier texte (**manager.txt**).
6. Enregistrer les informations du Manager « Imane » (lastName, firstName, baseSalary, respBonus) dans un fichier texte (**manager.txt**).
7. Parcourir le fichier **manager.txt** pour afficher les managers.