TP Noté 2008-2009 - Durée : 2h

POO : Programmation Orientée Objet

Première année

Supports et notes de Cours/TD/TP autorisés.

Le but du TP est d'implémenter une mini-base de données d'un service de ressource humaine.

Élements fournis. Vous trouverez dans le répertoire `/home/depot/1A/POO/TP_NOTE` un ensemble de classes de test. Recopiez ces classes dans votre répertoire de travail, à savoir le répertoire : `/home/depot/TP-NOTES/POO-<horaire>/<login>`. L'accès à ce répertoire sera automatiquement interdit à la fin de la séance.

Tests et Compilation. Les classes de test fournies vous permetteront de tester les différentes classes que vous implémenterez. Ces classes vous afficheront quels sont les tests qui ne s'exécutent pas correctement et vous donneront à titre indicatif un score pouvant s'apparenter à votre note de TP. **À la fin de la séance de TP, il est impératif que toutes vos classes compilent ! Un manquement à cette règle sera sévérement sanctionné**.
Pour compiler votre programme et notamment les tests fournis vous devrez utiliser la version 1.6 de java. Si vous n'avez jamais réalisé cette manipulation au cours des séances de travaux pratiques précédentes, vous trouverez un script nommé `env16.sh` qu'il vous suffira d'exécuter (par la commande `source ./env16.sh`) avant de lancer pour la première fois le compilateur javac.

⋆ ⋆ ⋆

▶ **Question 1.** L'interface `people.Person`.
Implémentez une interface `people.Person` représentant le comportement général d'une personne :
- une méthode `getName()` sans paramètre et dont le type de retour est une chaîne de caractères ;
- une méthode `setAge()` sans type de retour, prenant un paramètre de type `int` ;
- une méthode `getAge()` sans paramètre et dont le type de retour est une valeur de type `int` ;
- une méthode `getUniqueId()` dont le type de retour est une chaîne de caractères.

▶ **Question 2.** La classe abstraite `people.PersonImpl`.
Écrivez une classe *abstraite* `people.PersonImpl` implémentant partiellement l'interface `people.Person`. Cette classe doit :
- conserver l'âge et le nom de la personne ;
- définir les méthodes de l'interface `people.Person`.
- ne pas définir la méthode `getUniqueId()` qui sera abstraite (pour être définie dans les sous-classes de cette classe) ;
- redéfinir la méthode `toString()`. Pour afficher les informations d'une personne de la manière suivante : `"[identifiantUnique] nomdelapersonne (age: 27)"`
- redéfinir la méthode `equals()`. Pour tester l'égalité ;
- fournir un constructeur prenant en paramètre le nom et l'âge de la personne ;
- fournir un constructeur de copie permettant de créer une copie de la personne passée en paramètre.

✓ **Validation 1.** Testez votre implémentation.
Compilez et exécutez la classe de test founie nommé `people.TestPerson`. Si nécessaire, corriger les différentes erreurs dans votre code.

▶ **Question 3.** L'interface `people.Employee`.
Implémentez une interface `people.Employee` héritant de l'interface `people.Person` qui modélise de manière générale un employé :
- une méthode `getUniqueId()` sans paramètre dont le type de retour est une chaîne de caractères ;

- une méthode `getSalary()` sans paramètre et dont le type de retour est une valeur de type `double` ;
- une méthode `setSalary()` sans type de retour, prenant en paramètre un salaire de type `double`.

▶ **Question 4.** La classe `people.EmployeeImpl`.
Écrivez une classe `people.EmployeeImpl` héritant de la classe `people.PersonImpl` et implémentant l'interface `people.Employee`. Cette classe doit :

- conserver le salaire de l'employé ;
- conserver le NUMEN (NUMéro Education Nationale) de l'employé qui est chaîne de caractères qui sert à identifier de manière unique un employé ;
- définir la méthode abstraite de la classe `people.PersonImpl` ;
- définir les méthodes de l'interface `people.Employee` ;
- redéfinir la méthode `toString()`. Pour afficher les informations d'un employé de la manière suivante : `"[NUMEN] nomdelapersonne (age: 27) salary: 1900 euros"`
- redéfinir la méthode `equals()`. Pour tester l'égalité ;
- fournir un constructeur prenant en paramètre le NUMEN, le nom, l'âge de l'employé et son salaire ;
- fournir un constructeur de copie permettant de créer une copie de l'employé donné en paramètre.

✓ **Validation 2.** Testez votre implémentation.
Compilez et exécutez la classe de test fournie nommée `people.TestEmployee`. Si nécessaire, corriger les différentes erreurs dans votre code.

▶ **Question 5.** L'interface `people.Student`.
Implémentez une interface `people.Student` héritant de l'inteface `people.Person` qui représente le comportement général d'un étudiant :

- une méthode `getUniqueId()` sans paramètre dont le type de retour est une chaîne de caractères ;
- une méthode `getGrade()` sans paramètre et dont le type de retour est une valeur de type `int` ;
- une méthode `setGrade()` sans type de retour, prenant un paramètre un grade de type `int` ;
- trois constantes entières de type `int` : `YEAR_1ST`, `YEAR_2ND` et `YEAR_3RD` représentant les trois années de l'école.

▶ **Question 6.** La classe `people.StudentImpl`.
Écrivez une classe `people.StudentImpl` héritant de la classe `people.PersonImpl` et implémentant l'interface `people.Student`. Cette classe doit :

- conserver le grade (en quelle année il est inscrit : 1, 2 ou 3) ;
- conserver le numéro d'étudiant, chaîne de caractères qui sert à identifier de manière unique un étudiant ;
- définir la méthode abstraite de la classe `people.PersonImpl` ;
- définir les méthodes de l'interface `people.Student` ;
- redéfinir la méthode `toString()`. Pour afficher les informations d'un étudiant de la manière suivante : `"[NUMEN] nomdelapersonne (age: 27) grade: 1A"` si l'étudiant et de grade 1, `"[NUMEN] nomdelapersonne (age: 27) grade: 2A"` si l'étudiant et de grade 2, ...
- redéfinir la méthode `equals()`. Pour tester l'égalité ;
- fournir un constructeur prenant en paramètre le numéro d'étudiant, le nom, l'âge et le grade de l'étudiant ;
- fournir un constructeur de copie permettant de créer une copie de l'étudiant donné en paramètre.

✓ **Validation 3.** Testez votre implémentation.
Compilez et exécutez la classe de test fournie nommée `people.TestStudent`. Si nécessaire, corriger les différentes erreurs dans votre code.

▶ **Question 7.** L'interface `people.PeopleDatabase`.
Implémentez une interface `people.PeopleDatabase` représentant le comportement général d'une base de données de personnes :

- une méthode `add()` sans type de retour qui prend un paramètre de type `Person` ;
- une méthode `get()` prenant une chaîne de caractères (un identifiant) en paramètre et retournant l'objet de type `people.Person` correspondant à cet identifiant. Si un tel objet n'est pas trouvé, la méthode retournera une référence `null` ;
- une méthode `getPersonCount()` sans paramètre dont la valeur de retour est de type `int` correspondant au nombre de personnes enregistrées dans la base ;
- une méthode `getEmployeeCount()` sans paramètre dont la valeur de retour est de type `int` correspondant au nombre d'employés enregistrés dans la base ;
- une méthode `getStudentCount()` sans paramètre dont la valeur de retour est de type `int` correspondant au nombre d'étudiants enregistrés dans la base ;
- une méthode `getStudentAverageAge()` sans paramètre dont la valeur de retour est de type `double` égale à l'âge moyen des étudiants de la base ;
- une méthode `getAverageSalary()` sans paramètre dont la valeur de retour est de type `double` égale au salaire moyen des employés ;
- une méthode `getOldestPerson()` sans paramètre dont la valeur de retour est de type `people.Person` correspondant à la personne la plus âgée de la base. Si plusieurs personnes ont le même âge, alors la *première* personne trouvée avec cet âge doit être retournée ;
- une méthode `getRichestEmployee()` sans paramètre dont la valeur de retour est de type `people.Employee` correspondant à l'employé ayant le salaire le plus élevé. Si plusieurs personnes ont le même salaire (celui le plus élevé), alors le *dernier* employé trouvé ayant le salaire le plus élevé doit être retourné.

► **Question 8.** La classe `people.PeopleDatabaseFixedImpl`.
Écrivez une classe `people.PeopleDatabaseFixedImpl` implémentant l'interface `people.PeopleDatabase`. Cette classe doit :

- conserver les personnes enregistrées dans la base sous la forme d'un tableau à taille fixe de références de type `people.Person` ;
- définir les méthodes de l'interface `people.PeopleDatabase` ;
- fournir un constructeur prenant en paramètre le nombre maximal de personnes que l'on peut enregistrer dans la base.

✓ **Validation 4.** Testez votre implémentation.
Compilez et exécutez la classe de test fournie nommée `people.TestPeopleDatabaseFixed`. Si nécessaire, corriger les différentes erreurs dans votre code.

► **Question 9.** La classe `people.PeopleDatabaseDynamicImpl`.
Écrivez une classe `people.PeopleDatabaseDynamicImpl` implémentant l'interface `people.PeopleDatabase`. Cette classe doit :

- conserver les personnes enregistrées dans la base sous la forme d'un tableau à taille variable (utilisez la classe `java.util.ArrayList`) de type `people.Person` ;
- définir les méthodes de l'interface `people.PeopleDatabase` ;
- fournir un constructeur sans paramètre.

✓ **Validation 5.** Testez votre implémentation.
Compilez et exécutez la classe de test fournie nommée `people.TestPeopleDatabaseDynamic`. Si nécessaire, corriger les différentes erreurs dans votre code.

✓ **Validation 6.** Testez votre implémentation complète.
Afin de relancer l'ensemble des tests fournis, vous pouvez compiler et exécuter la classe de test nommée `people.TestAll`.

Bonne chance ;-)