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.

Réponse

people.Person

```
1  package people;
2
3  public interface Person {
4
5          public String getName();
6
7          public void setAge(int age);
8
9          public int getAge();
10
11         public String getUniqueId();
12
13 }
```

Fin réponse

▶ **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.

Réponse

people.PersonImpl

```
package people;

public abstract class PersonImpl implements Person {

	protected String name;

	protected int age;

	public PersonImpl(String name, int age) {
		this.name = name;
		this.age = age;
	}

	public PersonImpl(PersonImpl other) {
		this.name = other.name;
		this.age = other.age;
	}

	public String getName() {
		return this.name;
	}

	public void setAge(int age) {
		this.age = age;
	}

	public int getAge() {
		return this.age;
	}

	public abstract String getUniqueId();

	@Override
	public String toString() {
		return "[" + this.getUniqueId() + "] " + this.name + " (age: " + this.age + ")";
	}

	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + age;
		result = prime * result + ((name == null) ? 0 : name.hashCode());
		return result;
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		PersonImpl other = (PersonImpl) obj;
		if (age != other.age)
			return false;
		if (name == null) {
			if (other.name != null)
				return false;
		} else if (!name.equals(other.name))
			return false;
		return true;
	}

}
```

Fin réponse

✓ **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`.

Réponse

people.Employee

```
1  package people;
2
3  public interface Employee extends Person {
4
5          public String getUniqueId();
6
7          public void setSalary(double salary);
8
9          public double getSalary();
10
11 }
```

Fin réponse

▶ **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.

Réponse

people.EmployeeImpl

```
1  package people;
2
3  public class EmployeeImpl extends PersonImpl implements Employee {
4
5          private String numen;
6
7          private double salary;
8
9          public EmployeeImpl(String numen, String name, int age, double salary) {
10                 super(name, age);
11                 this.numen = numen;
12                 this.salary = salary;
13         }
14
15         public EmployeeImpl(EmployeeImpl other) {
16                 super(other);
17                 this.numen = other.numen;
18                 this.salary = other.salary;
19         }
20
21         @Override
22         public String getUniqueId() {
23                 return this.numen;
24         }
25
26         public void setSalary(double salary) {
27                 this.salary = salary;
28         }
29
30         public double getSalary() {
31                 return this.salary;
32         }
33
34         @Override
35         public String toString() {
36                 return super.toString() + " salary: " + this.salary + " euros";
37         }
38
39         @Override
40         public int hashCode() {
41                 final int prime = 31;
42                 int result = super.hashCode();
43                 result = prime * result + ((numen == null) ? 0 : numen.hashCode());
44                 long temp;
45                 temp = Double.doubleToLongBits(salary);
46                 result = prime * result + (int) (temp ^ (temp >>> 32));
47                 return result;
```

```
48          }
49
50          @Override
51          public boolean equals(Object obj) {
52                  if (this == obj)
53                          return true;
54                  if (!super.equals(obj))
55                          return false;
56                  if (getClass() != obj.getClass())
57                          return false;
58                  EmployeeImpl other = (EmployeeImpl) obj;
59                  if (numen == null) {
60                          if (other.numen != null)
61                                  return false;
62                  } else if (!numen.equals(other.numen))
63                          return false;
64                  if (Double.doubleToLongBits(salary) != Double.doubleToLongBits(other.salary))
65                          return false;
66                  return true;
67          }
68
69  }
```

Fin réponse

✓ **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.

Réponse

people.Student

```
1   package people;
2
3   public interface Student extends Person {
4
5           public static final int YEAR_1ST = 1;
6           public static final int YEAR_2ND = 2;
7           public static final int YEAR_3RD = 3;
8
9           public abstract String getUniqueId();
10
11          public abstract void setGrade(int grade);
12
13          public abstract int getGrade();
14
15  }
```

Fin réponse

► **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.

Réponse

people.StudentImpl

```
1  package people;
2
3  public class StudentImpl extends PersonImpl implements Student {
4
5          private String studentId;
6
7          private int grade;
8
9          public StudentImpl(String studentId, String name, int age, int grade) {
10                 super(name, age);
11                 this.studentId = studentId;
12                 this.grade = grade;
13         }
14
15         public StudentImpl(StudentImpl other) {
16                 super(other);
17                 this.studentId = other.studentId;
18                 this.grade = other.grade;
19         }
20
21         @Override
22         public String getUniqueId() {
23                 return this.studentId;
24         }
25
26         public void setGrade(int grade) {
27                 this.grade = grade;
28         }
29
30         public int getGrade() {
31                 return this.grade;
32         }
33
34         @Override
35         public String toString() {
36                 StringBuffer buf = new StringBuffer();
37                 buf.append(super.toString());
38                 buf.append(" grade: ");
39                 switch (this.grade) {
40                 case Student.YEAR_1ST:
41                         buf.append("1A");
42                         break;
43                 case Student.YEAR_2ND:
44                         buf.append("2A");
45                         break;
46                 case Student.YEAR_3RD:
47                         buf.append("3A");
48                         break;
49                 }
50                 return buf.toString();
51         }
52
53         @Override
54         public int hashCode() {
55                 final int prime = 31;
56                 int result = super.hashCode();
57                 result = prime * result + grade;
58                 result = prime * result + ((studentId == null) ? 0 : studentId.hashCode());
59                 return result;
60         }
61
62         @Override
63         public boolean equals(Object obj) {
64                 if (this == obj)
65                         return true;
66                 if (!super.equals(obj))
67                         return false;
68                 if (getClass() != obj.getClass())
69                         return false;
70                 StudentImpl other = (StudentImpl) obj;
71                 if (grade != other.grade)
72                         return false;
73                 if (studentId == null) {
74                         if (other.studentId != null)
75                                 return false;
76                 } else if (!studentId.equals(other.studentId))
77                         return false;
78                 return true;
79         }
80
81 }
```

Fin réponse

✓ **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.

Réponse

people.PeopleDatabase

```
1  package people;
2
3  public interface PeopleDatabase {
4
5          public void add(Person p);
6
7          public Person get(String id);
8
9          public int getPersonCount();
10
11         public int getEmployeeCount();
12
13         public int getStudentCount();
14
15         public double getStudentAverageAge();
16
17         public double getAverageSalary();
18
19         public Person getOldestPerson();
20
21         public Employee getRichestEmployee();
22
23 }
```

Fin réponse

▶ **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.

Réponse

people.PeopleDatabaseFixedImpl

```
1  package people;
2
3  public class PeopleDatabaseFixedImpl implements PeopleDatabase {
4
5          private Person[] persons;
6          private int personsCount;
7
8          public PeopleDatabaseFixedImpl(int initialSize) {
9                  this.persons = new Person[initialSize];
10                 this.personsCount = 0;
11         }
12
13         public void add(Person p) {
14                 if (this.personsCount < this.persons.length) {
15                         this.persons[personsCount++] = p;
16                 }
17         }
18
19         public Person get(String id) {
20                 for (int i = 0; i < this.personsCount; i++) {
21                         if (this.persons[i].getUniqueId().equals(id)) {
22                                 return this.persons[i];
23                         }
24                 }
25                 return null;
26         }
27
28         public int getPersonCount() {
29                 return this.personsCount;
30         }
31
32         public int getEmployeeCount() {
33                 int count = 0;
34
35                 for (int i = 0; i < this.personsCount; i++) {
36                         if (this.persons[i] instanceof Employee) {
37                                 count++;
38                         }
39                 }
40                 return count;
41         }
42
43         public int getStudentCount() {
44                 int count = 0;
45
46                 for (int i = 0; i < this.personsCount; i++) {
47                         if (this.persons[i] instanceof Student) {
48                                 count++;
49                         }
50                 }
51                 return count;
52         }
53
54         public double getAverageSalary() {
55                 int count = 0;
56                 int sum = 0;
57
58                 for (int i = 0; i < this.personsCount; i++) {
59                         if (this.persons[i] instanceof Employee) {
60                                 sum += ((Employee) this.persons[i]).getSalary();
61                                 count++;
62                         }
63                 }
64
65                 if (count == 0)
66                         return 0.;
67                 else
68                         return ((double) sum) / count;
69         }
70
71         public double getStudentAverageAge() {
72                 int count = 0;
73                 int sum = 0;
74
75                 for (int i = 0; i < this.personsCount; i++) {
76                         if (this.persons[i] instanceof Student) {
77                                 sum += this.persons[i].getAge();
78                                 count++;
79                         }
80                 }
81                 if (count == 0)
82                         return 0.;
83                 else
84                         return ((double) sum) / count;
85         }
86
87         @Override
88         public Person getOldestPerson() {
89                 Person oldestPerson = null;
90
91                 if (this.personsCount > 0) {
92                         oldestPerson = this.persons[0];
93
94                         for (int i = 1; i < this.personsCount; i++) {
```

```
95                                  if (this.persons[i].getAge() > oldestPerson.getAge()) {
96                                          oldestPerson = this.persons[i];
97                                  }
98                          }
99                  }
100
101                 return oldestPerson;
102         }
103
104         @Override
105         public Employee getRichestEmployee() {
106                 Employee richestEmployee = null;
107                 int index = 0;
108
109                 while (richestEmployee == null && index < this.personsCount) {
110                         if (this.persons[index] instanceof Employee) {
111                                 richestEmployee = (Employee) this.persons[index];
112                         }
113                         index++;
114                 }
115
116                 for (int i = index; i < this.personsCount; i++) {
117                         if (this.persons[i] instanceof Employee
118                                         && ((Employee) this.persons[i]).getSalary() >= richestEmployee.getSalary()) {
119                                 richestEmployee = (Employee) this.persons[i];
120                         }
121                 }
122
123                 return richestEmployee;
124         }
125
126 }
```

Fin réponse

✓ **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.

Réponse

people.PeopleDatabaseDynamicImpl

```
1  package people;
2
3  import java.util.ArrayList;
4
5  public class PeopleDatabaseDynamicImpl implements PeopleDatabase {
6
7          private ArrayList<Person> persons;
8
9          public PeopleDatabaseDynamicImpl() {
10                 this.persons = new ArrayList<Person>();
11         }
12
13         public void add(Person p) {
14                 this.persons.add(p);
15         }
16
17         public Person get(String id) {
18                 for (Person p : this.persons) {
19                         if (p.getUniqueId().equals(id))
20                                 return p;
21                 }
22                 return null;
23         }
24
25         public int getPersonCount() {
26                 return this.persons.size();
27         }
28
29         public int getEmployeeCount() {
30                 int count = 0;
31
32                 for (Person p : this.persons) {
33                         if (p instanceof Employee) {
```

```
34                                      count++;
35                              }
36                      }
37
38                      return count;
39              }
40
41              public int getStudentCount() {
42                      int count = 0;
43
44                      for (Person p : this.persons) {
45                              if (p instanceof Student) {
46                                      count++;
47                              }
48                      }
49
50                      return count;
51              }
52
53              public double getAverageSalary() {
54                      int count = 0;
55                      int sum = 0;
56
57                      for (Person p : this.persons) {
58                              if (p instanceof Employee) {
59                                      sum += ((Employee) p).getSalary();
60                                      count++;
61                              }
62                      }
63
64                      if (count == 0)
65                              return 0.;
66                      else
67                              return ((double) sum) / count;
68              }
69
70              public double getStudentAverageAge() {
71                      int count = 0;
72                      int sum = 0;
73
74                      for (Person p : this.persons) {
75                              if (p instanceof Student) {
76                                      sum += p.getAge();
77                                      count++;
78                              }
79                      }
80
81                      if (count == 0)
82                              return 0.;
83                      else
84                              return ((double) sum) / count;
85              }
86
87              @Override
88              public Person getOldestPerson() {
89                      Person oldestPerson = null;
90
91                      if (this.persons.size() > 0) {
92                              oldestPerson = this.persons.get(0);
93
94                              Person currentPerson;
95                              for (int i = 1; i < this.persons.size(); i++) {
96                                      currentPerson = this.persons.get(i);
97                                      if (currentPerson.getAge() > oldestPerson.getAge()) {
98                                              oldestPerson = currentPerson;
99                                      }
100                             }
101                     }
102
103                     return oldestPerson;
104             }
105
106             @Override
107             public Employee getRichestEmployee() {
108                     Employee richestEmployee = null;
109                     int index = 0;
110
111                     while (richestEmployee == null && index < this.persons.size()) {
112                             if (this.persons.get(index) instanceof Employee) {
113                                     richestEmployee = (Employee) this.persons.get(index);
114                             }
115                             index++;
116                     }
117
118                     for (int i = index; i < this.persons.size(); i++) {
119                             if (this.persons.get(i) instanceof Employee
120                                             && ((Employee) this.persons.get(i)).getSalary() >= richestEmployee.getSalary()) {
121                                     richestEmployee = (Employee) this.persons.get(i);
122                             }
123                     }
124
125                     return richestEmployee;
126             }
127
128     }
```

Fin réponse

✓ **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 ;-)

CORRECTION