TD - Séance n°3 - Correction

Héritage

1 Héritage

Exercice 1 On définit les classes `A`,`B`,`C` de la manière suivante :

```
1  public class A {
2    public void g() {
3      System.out.println(0);
4    }
5  }
6  public class B extends A {
7    public void g() {
8      System.out.println(1);
9    }
10 }
11 public class C extends A {}
```

1. Pourquoi n'a-t-on pas besoin de definir les constructeurs ?

Correction : Car le constructeur par defaut pour chacune des 3 classes est ci-dessus :

```
1  public class A {
2    public A() {} // super() implicite (A extends Object,
       though implicitly)
3    ...
4  }
5  public class B extends A {
6    public B() {} // super() implicite
7    ...
8  }
9  public class C extends A {
10   public C() {} // super() implicite
11   ...
12 }
```

On écrit un programme de test qui contient le code suivant :

```
1  public class Test {
2    public static void main(String[] args) {
3      A[] tab = new A[3];
4      tab[0] = new A();
5      tab[1] = new B();
6      tab[2] = new C();
7      for (int i=0; i<3; i++) { tab[i].g(); }
8    }
9  }
```

2. Qu'affiche l'exécution de ce programme ?

Correction : Réponse :

```
0
1
0
```

`tab[0]` est un objet de type `A`. `tab[1]` est un objet de type `B`. Comme `B` étend `A` et redéfinit la méthode `g()`, lorsqu'on appelle `g()` sur cet objet, la méthode de `B` sera utilisée. `tab[2]` est un objet de type `C`. Comme `C` étend `A` mais ne redéfinit pas la méthode `g()`, la méthode `g()` héritée de `A` sera utilisée.

3. Parmi les instructions suivantes, lesquelles feront une erreur à la compilation ?

```
    B o1 = new A();
    A o2 = new B();
    B o3 = new C();
    Object o4 = new B();
```

Correction : Réponse : `o1` et `o3` feront une erreur. Plus en détail :
- `B o1 = new A();` : Erreur de compilation car on essaie d'assigner une instance de la classe `A` à une référence de type `B`. Cependant, `A` n'est pas une sous-classe de `B`, donc cette conversion n'est pas permise.
- `A o2 = new B();` : Pas d'erreur de compilation. `B` est une sous-classe de `A`, donc une instance de `B` peut être assignée à une référence de type `A`.
- `B o3 = new C();` : Erreur de compilation car `C` n'est pas une sous-classe de `B`. Même si `C` et `B` sont toutes les deux des sous-classes de `A`, il n'existe pas de relation d'héritage directe entre `B` et `C`.
- `Object o4 = new B();` : En Java, toutes les classes sont des sous-classes de `Object`, donc une instance de `B` peut être assignée à une référence de type `Object`.

4. Est-il correct d'écrire `(new B()).toString()` ? Si oui, d'où la méthode `toString()` vient-elle ?

Correction : Réponse : Oui. `toString` est héritée de `Object` qui est la superclasse de toutes les classes en `Java`. Par conséquent, toute instance de n'importe quelle classe en `Java` héritera automatiquement de la méthode `toString()`.

2 Modélisation

Exercice 2 On définit une classe `Personne` de la manière suivante :

```
public class Personne {
  private String name;
  public Personne(String name) {
    this.name = name;
  }
}
```

On veut ici illustrer la notion d'héritage en modélisant la structure de la société française au moyen-âge. Cette structure reposait sur une division en

trois ordres : la noblesse (les nobles), le clergé (les prêtres) et le tiers-état (les roturiers). Un membre de la noblesse est un noble, un membre du clergé est un prêtre, et un membre du tiers-état est un roturier.

1. Définir une méthode `toString` dans la classe `Personne`, de sorte que l'exécution du code suivant produise :
   Je m'appelle Louis.

```
public static void main(String args[]) {
  Personne p = new Personne("Louis");
  System.out.println(p);
}
```

Correction :

```
  public String toString() {
    return "Je m'appelle " + this.name + ". ";
  }
```

2. Définir des classes `Noble`, `Pretre` et `Roturier` qui héritent de `Personne`. Dans ces classes, redéfinir `toString` en utilisant un appel à `super.toString()`, de telle sorte que l'exécution du code suivant produise :
   Je m'appelle Louis. Je suis un noble.

```
public static void main(String args[]) {
  Personne n = new Noble("Louis");
  System.out.println(n);
}
```

Correction :

```
public class Noble extends Personne {
  public Noble(String name) {
    super(name);
  }

  public String toString() {
    return super.toString() + "Je suis un noble.";
  }
}
```

```
public class Pretre extends Personne {
  public Pretre(String name) {
    super(name);
  }
  public String toString() {
    return super.toString() + "Je suis un prêtre.";
  }
}
```

```
public class Roturier extends Personne {
  public Roturier(String name) {
    super(name);
```

```
  }
  public String toString() {
    return super.toString() + "Je suis un roturier.";
  }
}
```

3. On ajoute maintenant à la classe `Personne` :

```
private int argent = 0;
public void recevoirArgent(int i) {
  this.argent += i;
}
```

Ajouter une méthode `boolean donnerArgent(int i)`, renvoyant `false` lorsque la personne n'a pas assez d'argent sur elle.

Correction :

```
  public boolean donnerArgent(int i) {
    if (this.argent < i) {
      return false;
    }
    this.argent -= i;
    return true;
  }
```

4. On considère maintenant que les nobles ont le droit de contracter des dettes (avoir une somme d'argent négative). Que doit-on rajouter au code, et où ?

Correction : `argent` étant un attribut privé de `Personne`, il n'est pas accessible par `Noble`. Il faut un accesseur et un mutateur public dans la classe `Personne` avant de redéfinir la méthode `donnerArgent`.

`Personne.java` :

```
  public void setArgent(int i) {
    this.argent = i;
  }

  public int getArgent() {
    return this.argent;
  }
```

`Noble.java` :

```
  public boolean donnerArgent(int i) {
    setArgent(getArgent() - i);
    return true;
  }
```

5. Le tiers-état est un ordre très hétérogène socialement, qui comprend à la fois des paysans, des artisans et des bourgeois. Quelles classes doit-on créer pour modéliser cela ? De quelle classe doivent-elles hériter ?

Correction : On crée les classes `Paysan`, `Artisan`, `Bourgeois`, qui héritent de `Roturier`.

6. On considère maintenant une classe `Societe`, qui a comme attribut un tableau de `Personne`.
   — Écrire un constructeur de `Societe`, qui prend en argument un entier n et qui crée une société de n personnes, de rôle social choisi aléatoirement. Le nom de la i-ème personne de la societé peut être `Personne_i`.
   On peut utiliser `Math.random()` (qui se trouve dans le package `java.lang`), qui renvoie un `double` entre `0.0` (inclus) et `1.0` (exclus), ou un objet de la classe `Random` (qui se trouve dans le package `java.util`), dont une utilisation possible est la suivante :

```
1  Random r = new Random();
2  int i = r.nextInt(5); // i est pris dans
       {0,1,2,3,4}
```

Correction :

```
1  import java.util.Random;
2
3  public class Societe {
4    private Personne[] personnes;
5
6    public Societe(int n) {
7      double intervalle[] = {.1, .2, .5, .8};
8      personnes = new Personne[n];
9      for (int i = 0; i < n; i++) {
10       double role = Math.random();
11       Random r = new Random();
12       int argent = r.nextInt(100);
13       if (role <= intervalle[0]) {
14         personnes[i] = new Noble("Noble " + i);
15       }
16       else if (role <= intervalle[1]) {
17         personnes[i] = new Pretre("Pretre " + i);
18       }
19       else if (role <= intervalle[2]) {
20         personnes[i] = new Paysan("Paysan " + i);
21       }
22       else if (role <= intervalle[3]) {
23         personnes[i] = new Artisan("Artisan " + i);
24       }
25       else {
26         personnes[i] = new Bourgeois("Bourgeois " +
               i);
27       }
28       personnes[i].setArgent(argent);
29     }
30   }
```

   — Implémenter une méthode `toString()` qui accumule les `toString()` de chaque membre de la société.
   Correction : `Societe.java` :

```java
public String toString() {
    String res = "";
    for (int i = 0; i < personnes.length; i++) {
      res += personnes[i].toString() + "\n";
    }
    return res;
}
```

— Écrire une méthode : `public int argentTotal()`, qui renvoie la somme de l'argent que chaque membre de la société possède.

Correction : `Societe.java` :

```java
public int argentTotal() {
  int res = 0;
  for (int i = 0; i < personnes.length; i++) {
    res += personnes[i].getArgent();
  }
  return res;
}
```

Exercice 3 On va maintenant légèrement changer l'implémentation de `Societe` pour qu'elle évolue avec de nouvelles naissances et des morts des membres de la société. On va pour cela ajouter un champ `age` à la classe `Personne`, et le tableau de `Personne` de la classe `Societe` va devenir une `LinkedList<Personne>`.

Correction : Changer le contructeur de la classe `Societe` et tout autre part affecté.

1. Ajouter une méthode `boolean anniversaire()` à la classe `Personne` faisant vieillir une personne d'un an et renvoyant `true`.
2. Redéfinir `anniversaire()` pour les classes héritées de `Personne` de façon à ce que la méthode renvoie `false` si l'individu a atteint un âge trop avancé (on donnera des espérances de vie différentes aux différentes classes).
3. Ajouter à la classe `Personne` une méthode `Personne enfanter()` renvoyant `null` et la redéfinir dans les différentes classes de façon à ce que cette méthode :
   — Renvoie `null` si l'individu est trop jeune.
   — Renvoie une nouvelle personne sinon. Le type de cette personne dépend de la classe dont on appelle la méthode (un noble engendrera un noble ou un prêtre, un prêtre sera sans enfant, et les roturiers engendre des prêtres ou des roturiers, de professions variées)
4. Ajouter une méthode `void anniversaire()` à la classe `Societe` qui fait s'écouler un an. Pendant cette année :
   — Chaque membre de la société vieillit d'un an. Ayant atteint son âge limite, la personne est retirée de la société.
   — Chaque membre adulte de la société a une petite chance d'engendrer un enfant, ajouté à la société.

5. Dans la méthode `main`, initialisez une société de 100 personnes, et faites écouler 50 ans en affichant pour chaque année écoulée le nombre de personnes dans la société.

```
public class Test{
  public static void main(String[] args){
    ...
  }
}
```

Correction : Dans `Societe.java`, il faut utiliser `add` pour ajouter à la liste de personnes, `size` pour obtenir la taille, et `get` pour accéder à un élément.

```
import java.util.Random;
import java.util.LinkedList;

public class Societe {
  private LinkedList<Personne> personnes;

  public Societe(int n) {
    double intervalle[] = {.1, .2, .5, .8};
    personnes = new LinkedList<Personne>();
    for (int i = 0; i < n; i++) {
      double role = Math.random();
      Random r = new Random();
      if (role <= intervalle[0]) {
        personnes.add(new Noble("Noble " + i));
      }
      else if (role <= intervalle[1]) {
        personnes.add(new Pretre("Pretre " + i));
      }
      else if (role <= intervalle[2]) {
        personnes.add(new Paysan("Paysan " + i));
      }
      else if (role <= intervalle[3]) {
        personnes.add(new Artisan("Artisan " + i));
      }
      else {
        personnes.add(new Bourgeois("Bourgeois " + i));
      }
    }
  }

  public LinkedList<Personne> getPersonnes() {
    LinkedList<Personne> cloned = new LinkedList<Personne
        >();
    for (int i = 0; i < personnes.size(); i++) {
      cloned.add(personnes.get(i));
    }
    return cloned;
  }

  public void anniversaire() {
```

```
    // we take the population size before the loop, such
    // that new children will not be taken into account
    int population_size_last_year = personnes.size();
    for (int i = 0; i < population_size_last_year; i++) {
      if (!personnes.get(i).anniversaire()) {
        personnes.remove(i);
        // the list will be shifted after the remove,
        // so we have to take this into account
        i--;
        population_size_last_year--;
      } else {
        boolean nouveau_enfant = (Math.random() <= .05);
        if (nouveau_enfant) {
          Personne enfant = personnes.get(i).enfanter();
          if (enfant != null) {
            personnes.add(enfant);
          }
        }
      }
    }
  }
}
```

Personne.java :

```
public class Personne {
  private String name;
  protected int age;
  private final int esperance_de_vie;
  public static final int age_majeur = 18;

  public Personne(String name, int esperance) {
    this.name = name;
    this.esperance_de_vie = esperance;
  }

  public boolean anniversaire() {
    if (this.age + 1 > esperance_de_vie) {
      return false;
    }
    this.age += 1;
    return true;
  }

  public Personne enfanter() {
    return null;
  }
}
```

Noble.java :

```
public class Noble extends Personne {
  public static final int esperance_de_vie = 90;

```

```
  public Noble(String name) {
    super(name, esperance_de_vie);
  }

  public Personne enfanter() {
    if (this.age < Personne.age_majeur) {
      return null;
    }
    double role = Math.random();
    if (role < .5) {
      return new Noble("");
    } else {
      return new Pretre("");
    }
  }
}
```

Pretre.java;

```
public class Pretre extends Personne {
  public static final int esperance_de_vie = 90;

  public Pretre(String name) {
    super(name, esperance_de_vie);
  }

  // un prêtre sera sans enfants donc on ne remplace
  // pas la fonction enfanter().
}
```

Roturier.java :

```
public class Roturier extends Personne {
  public Roturier(String name, int esperance) {
    super(name, esperance);
  }

  public Personne enfanter() {
    if (this.age < Personne.age_majeur) {
      return null;
    }
    double role = Math.random();
    if (role < .4) {
      return new Pretre("");
    } else if (role < .6) {
      return new Paysan("");
    } else if (role < .8) {
      return new Artisan("");
    } else {
      return new Bourgeois("");
    }
  }
}
```

Paysan.java :

```java
public class Paysan extends Roturier {
  public static final int esperance_de_vie = 60;

  public Paysan(String name) {
    super(name, esperance_de_vie);
  }
}
```

Artisan.java :

```java
public class Artisan extends Roturier {
  public static final int esperance_de_vie = 66;

  public Artisan(String name) {
    super(name, esperance_de_vie);
  }
}
```

Bourgeois.java :

```java
public class Bourgeois extends Roturier {
  public static final int esperance_de_vie = 75;

  public Bourgeois(String name) {
    super(name, esperance_de_vie);
  }
}
```

Test.java :

```java
public class Test {
  public static void main(String[] args) {
    Societe france = new Societe(100);
    for (int i = 0; i < 50; i++) {
      france.anniversaire();
      System.out.println("nombre de personnes : " + france.
          getPersonnes().size());
    }
  }
}
```