Université Paris Diderot **Programmation Orientée Objet**
L2, EIDD et M1 linguistiques **Année 2015-2016**

TD n° 4

Héritage et interfaces 2

Dans cet TD nous allons continuer à manipuler le concept d'héritage ainsi que les interfaces.

Exercice 1 On considère comme lors du précédent TD une classe Personne, avec des attributs légèrement modifiés :

```
1  public class Personne {
2
3  private String nom;
4  private int argent;
5  private int pdv; \\ les points de vie de la personne
6
7  public Personne(String nom, int argent, int pdv){
8      this.nom = nom;
9          this.argent = argent;
10         this.pdv = pdv;
11 }
12 public void gain(int n){
13         this.argent = this.argent + n;
14 }
15 public void perte(int n){
16         this.argent = this.argent - n;
17 }
18 public void blessure(int n){
19         this.pdv = this.pdv -n;
20 }
21 public String toString(){
22         return "Je m'appelle :" + this.nom + ". J'ai " + this.
               argent + " unités monétaires, et " + this.pdv " points
               de vie ".;
23 }
24 }
```

Comme dans le TD3, on a une classe `Noble` héritant de `Personne`. On rappelle qu'un noble possède un attribut `List<Roturier> roturiers`.

Dans cet exercice, nous allons modéliser l'une des activités favorite de la noblesse au Moyen-âge, à savoir la guerre. On créera pour cela une classe `Chevalier` (i.e les combattants de la noblesse) héritant de `Noble` ainsi que des classes `Archer` et `Fantassin` héritant de `Personne`. La classe `Fantassin` possède un attribut `int degat`. Les comportements militaires seront modélisés par une interface `Guerrier` contenant une méthode `void attaque(Personne p)` et devant être implémentée par `Chevalier`, `Archer` et `Fantassin`.

1. Faire une modélisation UML des classes (avec l'interface `Guerrier`) précédentes.
2. On va maintenant implémenter la méthode `attaque`. On rajoute pour cela un attribut `boolean estLibre = true` dans la classe `Chevalier`.

Implémenter la méthode `attaque` dans les classes `Chevalier`, `Archer` et `Fantassin`, sachant que

(a) un `Archer` tue la personne qu'il attaque (i.e il lui enlève tous ses points de vie),

(b) un `Chevalier` n'attaque une personne que si elle est elle-même une instance de `Chevalier`, et dans ce cas le `Chevalier` attaqué est capturé (i.e il perd sa liberté). Si on demande à un chevalier d'attaquer autre chose qu'un `Chevalier`, un message d'erreur s'affiche [1],

(c) un `Fantassin` capture la personne qu'il attaque si celle-ci est une instance de `Chevalier` et enlève `degat` aux points de vie de la personne si celle-ci n'est pas une instance de `Chevalier`.

3. Lorsqu'un chevalier est capturé, il a la possibilité de payer une rançon pour racheter sa liberté. Rajouter dans `Chevalier` une méthode `boolean rancon(int n, Personne p)` qui fonctionne de la façon suivante : si le chevalier capturé a les moyens de payer la rançon (i.e. son attribut `argent` est supérieur à `n`) alors il paye la rançon à `p` et regagne sa liberté. La méthode renvoie `true` si, et seulement si la rançon a été payée.

4. À partir du milieu du Moyen-âge, la guerre a changé et a vu l'apparition quasiment systématique de bandes de mercenaires (aussi appelées compagnies) dans les armées. Ces bandes étaient généralement menées par un chef, que l'on appelait condiottiere en Italie. Créer une classe `Condottiere` héritant de `Personne` et possédant comme attribut une liste de `Guerriers`.

Exercice 2 La guerre au Moyen-âge donnait rarement lieu à des batailles rangées mais consistait principalement en escarmouches et en pillages. Le but de cet exercice est de modéliser les pillages.

1. Créer une classe `Village` possédant comme attribut une liste de `Roturier`.

2. Créer une interface `Pillage` contenant une méthode `void attaque(Village v)`. Cette interface doit être implémentée par les classes `Chevalier` et `Condottiere`. Compléter la modélisation UML. La méthode `attaque` doit avoir le comportement suivant :

   (a) Lorsqu'un chevalier attaque un village, il capture tous les habitants (i.e il les rajoute dans la liste de ses roturiers),

   (b) Lorsqu'un `Condottiere` attaque un village, ce dernier est mis à sac (i.e chaque villageois se fait voler la moitié de son argent). Les gains récupérés sont répartis pour moitié entre le condottiere et pour l'autre moitié entre les membres de sa compagnie.

Exercice 3 On va maintenant donner un rôle (pacifique) aux clercs dans la guerre.

1. Rajouter à l'interface `Guerrier` une méthode `boolean reussite()`, destinée à tester si un attaque va réussir. En pratique :

1. La bonne modélisation consisterait à envoyer une exception.

- un `Chevalier` ou un `Fantassin` possède un attribut `double reussite = Math.random()`. La méthode `reussite()` tire un nombre compris entre 0 et 1, et renvoie `true` si l'attribut `reussite` lui est supérieur,
- un `Archer` réussit toujours son attaque.

Redéfinir la méthode `attaque` de la question 2 de l'exercice 1 de la façon suivante, de telle sorte que la méthode `attaque` n'agisse qu'en cas de réussite.

2. Créer une classe `Clerc` héritant de `Personne`. Les clercs ont la faculté de soigner les blessés. Rajouter un attribut `int soin` dans le classe `Clerc` ainsi qu'une méthode `void soigne(Personne p)`.
3. Écrire une classe `Pretre` qui hérite de `Clerc`. Lorsqu'un prêtre soigne quelqu'un, il le bénit également. On demande alors que l'attribut `reussite` de la personne soignée soit augmenté (mais reste bien sur inférieur à 1).