L3-CPOO
Travaux Pratiques

TP1 : Java & hiérarchie de classes

Exercice 1 : Résolution d'une équation du second degré

Réalisez un programme Java calculant la résolution d'une équation du second degré de la forme $ax^2 + bx + c = 0$. Pour cela, vous prendrez soin de produire 3 classes définissant les 3 cas possibles (0 solution, 1 solution et 2 solutions). C'est 3 classes utiliseront l'héritage, avec comme parent commun une classe de type `Binome` donnée comme suit :

```
public class Binome {
  // Donnees:
  protected double a,b,c,dis;

  // Methodes
  public Binome(double pa,double pb,double pc,double pdis)
  { a = pa;  b = pb;  c = pc;  dis = pdis;  }

  public Binome creation(double pa,double pb,double pc)
  { double delta;
    delta = pb * pb - 4.0 * pa * pc;
    if (delta < 0.0) {
      return new BinomeSol0(pa,pb,pc,delta);
    } else if (delta == 0.0) {
      return new BinomeSol1(pa,pb,pc,delta);
    } else {
      return new BinomeSol2(pa,pb,pc,delta);
    };
  }

  public void calculer_racine()
  { System.out.println(``Erreur si ici !'');  }

  public int nb_racine()
  { System.out.println(``Erreur si ici !'');  return 0;  }

  public double valeur_racine(int i)
  { System.out.println(``Erreur si ici !'');  return 0.0;  }
}
```

Exercice 2 : animaux

La classe *Animal* comporte deux sous-classes *Mammifère* et *Poisson* ; la classe *Mammifère* comporte elle-même deux nouvelles sous-classes : *Chien* et *Homme*. Modéliser et créer ces différentes classes en Java de telle sorte que la classe *TestAnimal* donnée comme suit :

```
public class TestAnimal {
  public static void main(String[] args) {
    Animal[] animaux = new Animal[5];
    animaux[0]=new Animal("Truc");;
    animaux[1]=new Animal();
    animaux[2]=new Chien("Medor");
    animaux[3]=new Homme() ;
    animaux[4]=new Homme ("Robert") ;
    for (int i=0; i<5; i++) {
       System.out.println(animaux[i].getType());
    }
  }
}
```

affiche le résultat suivant :

```
Je suis un animal de nom Truc.
Je suis un animal.
Je suis un animal de nom Medor. Je suis un mammifere. Je suis un chien.
Je suis un animal. Je suis un mammifere. Je suis un homme.
Je suis un animal de nom Robert. Je suis un mammifere. Je suis un homme.
```

Exercice 3 : péage autoroutier

Un service autoroutier a installé un péage sur une de ses routes principales. Dans une première version, chaque péage applique la même tarification à savoir :

- Les camions paient une taxe qui dépend du nombre d'essieux et de son poids total : la facturation est de 7 euros par essieu plus 15 euros par tonne.
- Les voitures paient une taxe fixe de 4 euros.

Pour cette version simplifiée,

- les occurrences de la classe *Voiture* ne nécessitent aucun attribut particulier de façon obligatoire ;
- les occurrences de la classe *Camion* possèdent 2 attributs privés *nbreEssieu*, *poidsTotal* (exprimé en tonnes) dont les valeurs sont restituées à l'aide des accesseurs *getNbreEssieu()* et *getPoidsTotal()* pour lesquels vous définirez les signatures adéquates ;
- chaque péage possède 2 attributs privés *nbreVehicule* et *totalCaisse* qui mémorisent le nombre de véhicules (camion ou voiture) ayant franchi le péage et le montant total des taxes perçues. Ces 2 variables sont mises à jour lors du passage d'un véhicule.

Pour réaliser cette application en Java, on vous demande de

- définir les classes *Péage*, *Voiture* et *Camion*. Il sera judicieux de définir une classe abstraite *Véhicule* ayant *Voiture* et *Camion* comme sous-classes et d'utiliser le typage dynamique (i.e. éviter les instructions conditionnelles sur le type des véhicules).

– définir une classe *Global* avec la méthode *main* permettant d'instancier des objets de type *Péage*, *Voiture* ou *Camion* et de simuler les passages des véhicule aux péages c'est-à-dire d'appeler les méthodes de mise à jour des attributs *nbreVehicule* et *totalCaisse* des péages concernés.

Variantes

– Le coût est variable suivant les péages : la facturation par essieu et tonne ou par voiture n'est plus identique à chaque péage mais spécifique à chacun d'entre eux.
– Des statistiques plus fines peuvent être proposées telles que remplacer *nbreVehicule* par deux attributs *nbreVoiture* et *nbreCamion* pour compter le nombre de voitures et de camions ayant franchi le péage.

Exercice 4 : des lapins

Créer une classe *Lapin*. Tout lapin possédera un attribut *vivant* permettant de savoir s'il est en vie ainsi qu'une méthode *passeALaCasserole()* qui le fera mourir. Pour éviter toute surpopulation, il devra être impossible de créer un nouveau lapin tant qu'il y a plus de 50 lapins vivants.

1. Dans une première version, vous pouvez, en cas de surpopulation créer un lapin sans vie (l'attribut *vivant* est initialisé à *false*).
   Il y a en fait, 3 cas de figures : (1) un lapin est vivant, (2) un lapin était vivant à un moment donné mais est mort à présent (il est passé à la casserole au cours de son existence), (3) un lapin n'a jamais pu être crée à cause de la surpopulation. Les 2 dernières situations ne peuvent être différenciées dans cette version.
   Expliquer pourquoi, on ne peut pas simplement renvoyer *null* en cas de surpopulation.

2. Dans une seconde version, rendre le constructeur *Lapin()* privé et créer un nouveau lapin à l'aide d'une méthode *Lapin creationLapin()* qui appelle le constructeur *Lapin()* s'il n'y a pas de surpopulation et renvoie un pointeur *null* sinon. Dans ce cas, les 3 situations peuvent être distinguées.

Remarques :

– Vous pouvez réduire la valeur de la population maximale autorisée pour vos tests.
– Une valeur *null* pour un pointeur est écrite en minuscules.