Université Paris Diderot **Programmation Orientée Objet**
L2, EIDD et M1 linguistiques **Année 2017-2018**

Contrôle Continu

Exercice 1 *Notions Fondamentales*

On considère les classes suivantes :

```
1  public class A{
2    public int x=2;
3    public int z=10;
4    public A(){
5      x=x*x;
6      }
7    public String h(A a){
8      return "h dans A";
9      }
10   public String g(){
11     return "g dans A";
12     }
13 }
```

```
1  public class B extends A{
2    public int y;
3    public int z=1000;
4    public B(int yy){
5    y=yy;
6    }
7    public void plus(int y){
8      y=y+y;
9    }
10   public String h(B b){
11     return "h dans B";
12   }
13   public String g(){
14     return "g dans B";
15   }
16   public static void main(String [] args){
17     A a = new A();
18     B b=new B(3);
19     A ab= new B(777);
20     B ba=(B) ab;
21
22     System.out.println(ab.z);
23     System.out.println(ba.z);
24
25     System.out.println(ab.g());
26     System.out.println(ba.g());
27
28     System.out.println(a.h(b));
29     System.out.println(b.h(a));
30
31     System.out.println(a.x);
32     b.plus(b.y)
33     System.out.println(b.y);
34   }
35 }
```

Écrivez et **expliquez** ce que l'exécution du code de classe B montrera dans l'écran.

Exercice 2 [Implémentation] On souhaite implémenter un habitat qui héberge des différents espèces des animaux. Pour rendre simple cette implémentation, les animaux seront bien seront herbivore ou carnivore. Il aura des méthodes qui contrôlent la faim de ces animaux et seulement les carnivores obtiendront un comportement particulière à l'intérieur de l'habitat. Les attributs seront toujours privés, mais vous pouvez utiliser des méthodes *getter()* et *setter()*.

1. Déclarez la classe **Animal** qui contient un entier faim et une chaîne de caractères espèce.
2. Ajoutez une méthode constructeur sans argument qui initialise les attributs de l'objet en 0 et "inconnue" et un constructeur qui accepte des paramètres pour l'initialisation de chaque attribut.
3. Déclarez les méthodes `void` unJourPasse( ) qui augmente l'attribut faim dde 1 et une autre `void` manger( ) que réduit faim de 2.
4. Ajoutez la méthode `void` trouverNourriture( ) qui estime aléatoirement si l'animal trouve de la nourriture. Dans le cas affirmatif, l'animal mangera, c'est-à-dire on utilise la méthode manger( ). *Remarque :* On peut utiliser la fonction `double Math.random()` qui retourne un nombre réel $0 \leq x < 1$.
5. Déclarez les classes enfants de la classe **Animal** : **Herbivore** et **Carnivore**.
6. Dans la classe carnivore, ajoutez un attribut boolean predateur et un constructeur avec arguments et un autre sans arguments qui initialise ce nouveau attribut en `false`.
7. On considère que les carnivores seront prédateurs si et seulement si faim est supérieur à 6. Redéfinissez les méthodes unJourPasse( ) et manger( ) pour qu'elles prennent en compte la règle de l'attribut predateur.
8. Déclarez la classe **Habitat** avec un seul attribut biotope : un tableau de la classe **Animal**. Ajoute un constructeur avec un seul argument entier n qui initialise l'objet comme un biotope qui a la place pour n animaux.
9. Ajoutez la méthode `boolean` migration(Animal [ ] migrateurs) qui retourne vrai dans le cas que les migrateurs puissent *tous* entrer dans l'habitat. En cas affirmatif, ajoutez chaque animaux du tableau migrateurs ou tableau biotope (Remarque, on souhaite ces animaux, *pas une copie* d'eux). Dans le cas contraire, la méthode retournera `false` et elle fera aucun changement.

   Remarque :Supposez que le biotope est vide.
10. Enfin, ajoutez la méthode `void` unJourPasse( ) qui ajoute en une unité la faim à chaque animal de l'habitat. Après, s'il existe un animal prédateur, il mangera le premier herbivore *vivant* (ce dernier deviendra `NULL`). Les autres animaux suivent la loi de trouverNourriture().

facultatif Si on veut compter tous les animaux créés, modifiez la classe **Animal** pour accomplir ce nouveau objectif.