L2 info S3
UE : Programmation Orientée Objets 1 - Java

TD2 - Héritage

Objectifs

A la fin de ce TD, vous devez être capables de :

1. Donner l'arborescence des classes.
2. Déterminer les attributs et méthodes des classes dérivées.
3. Faire la distinction entre redéfinition et surcharge de méthode.
4. Employer le mot clé `super` dans l'appel de constructeurs ou pour accéder aux membres (attributs, méthodes) du père.
5. Rappeler la règle des constructeurs : appel direct/indirect, explicite/implicite du constructeur père (avec `super` et `this`).
6. Faire des affectations en respectant la compatibilité classe parente-classe dérivée.
7. Appliquer le polymorphisme : pour une méthode écrire des appels sans errreur de compilation (typage statique) et déterminer celle qui sera effectivement exécutée (liaison dynamique).
8. Utiliser les modificateurs `final` pour une classe et `protected` pour un attribut.

1 Animal, oiseau et perroquet

Etudier les classes du programme donné dans la page suivante en faisant bien attention aux relations entre les types déclarés et les types courants.

Répondre aux questions suivantes :

1. Donner l'arborescence des classes.
2. Donner la super-classe de `Animal`.
3. Pour chacune des 3 classes (`Animal`, `Oiseau` et `Perroquet`), donner l'intégralité de leurs attributs et de leurs méthodes.
   Pour chaque méthode, préciser si elle est héritée, redéfinie ou spécifique à la classe.
4. Donner un exemple de surcharge (surdéfinition) de la méthode `viellir` dans une des classes dérivées (classe `Oiseau` ou `Perroquet`).
5. Donner un exemple de redéfinition de la méthode `viellir` dans une des classes dérivées.
6. Classe `Perroquet`,
   — dans le corps de son constructeur, qu'effectue l'instruction `super(s)` ?
   — dans la méthode `dire()`, qu'effectue l'instruction `super.dire()` ?
7. Donner l'affichage de ce programme.
8. Que se passe-t-il si on ajoute `o.apprendVole()` et `p.apprendParole()` avant la boucle `for` ?
9. De même, on voudrait faire voler *Birdy* et apprendre à parler à *Roger*. Pourquoi l'instruction suivante : `tab[3].apprendVole()` génère lors de la compilation du programme l'erreur suivante : `erreur tab[3] de type Animal ?` Comment modifier le code ? Pourquoi `o.apprendVole()` était acceptée sans aucune erreur ?

10. Si un oiseau est toujours un animal, comment vérifier qu'un animal est un oiseau ?
11. Compléter le programme pour que *Coco* possède toutes ses facultés.
12. Comment dire que la classe `Perroquet` ne peut plus être étendue ?

```
class Animal
{
  protected String nom ;
  protected int age = 0 ;

  public Animal(String s)
  {
    nom = s ;
  }

  public void vieillir()
  {
    age ++ ;
  }

  public void dire()
  {
     System.out.println("...");
  }
}
```

```
class Perroquet extends Oiseau
{
   protected boolean parle = false ;

   public Perroquet(String s)
   {
      super(s);
   }

   public void apprendParole()
   {
      parle = true ;
   }

   public void dire()
   {
      if (parle)
         System.out.println(nom);
      else
         super.dire();
   }
}
```

```
class Oiseau extends Animal
{
  protected boolean vole = false ;

  public Oiseau(String s)
  {
     super(s) ;
  }

  public void apprendVole()
  {
     vole = true ;
  }

  public void dire()
  {
    if (vole)
      System.out.println("cui-cui");
    else
      System.out.println("piou-piou");
  }
}
```

```
public class TestAnimaux
{
   public static void main(String[] args)
   {
      Animal a = new Animal("Joe");
      Oiseau o = new Oiseau("le piaf");
      Perroquet p = new Perroquet("Coco");
      Animal [] tab = new Animal[5];
      tab[0] = a;
      tab[1] = o ;
      tab[2] = p ;
      tab[3] = new Oiseau("Birdy");
      tab[4] = new Perroquet("Roger");
      for (int i = 0; i<5 ; i++)
      {
         tab[i].dire();
      }
   }
}
```

2 Villes et capitales

Soit le programme Java qui utilise les classes `Ville` et `Capitale`

```
public class TestVille
{
  public static void main(String args[])
  {
     Ville v1,v2;

     v1= new Ville ("Toulouse");
     v2 = new Ville ("Strasbourg", 272975);

     System.out.println(v1);
     System.out.println(v2);
     System.out.println();

     Capitale c1, c2;
     c1 = new Capitale("Paris", "France");
     c2 = new Capitale("Rome", "Italie", 2700000);
     c1.setNbHabitants(2181371);

     System.out.println(c1);
     System.out.println(c2);
     System.out.println();

     System.out.println("catégorie de la ville de " +
         v1.getNom() + " : " + v1.categorie());
     System.out.println("catégorie de la ville de " +
         v2.getNom() + " : " + v2.categorie());
     System.out.println("catégorie de la ville de " +
         c1.getNom() + " : " + c1.categorie());
     System.out.println();
  }
}
```

qui lors de l'exécution affichera le résultat suivant :

```
Ville de TOULOUSE.
Ville de STRASBOURG ; 272975 habitants.

Ville de PARIS ; 2181371 habitants. Capitale de FRANCE.
Ville de ROME ; 2700000 habitants. Capitale de ITALIE.

catégorie de la ville de TOULOUSE : ?
catégorie de la ville de STRASBOURG : A
catégorie de la ville de PARIS : C
```

Questions

Donner les instructions Java pour définir les classes `Ville` et `Capitale` en respectant les spécifications suivantes :

1. La classe `Ville` a
   les propriétés suivantes :
   - Une ville est décrite par deux informations : son nom et son nombre d'habitants. Les variables seront respectivement nommées `nom` et `nbHabitants` ; elles sont d'accès protégé (`protected`).
   - Le nom d'une ville ne peut varier ; il doit être connu dès l'instanciation de l'objet. Il est toujours représenté en majuscules.
     Rappel : dans la classe `String`, la méthode d'instance `String toUpperCase()` convertit tous les caractères de la chaîne en majuscule.
   - Le nombre d'habitants peut être inconnu ; sa valeur est alors 0. S'il est connu, il est toujours supérieur à zéro. Il peut varier pour un même objet `Ville` (suite par exemple à un nouveau recensement).

   les constructeurs
   - Un constructeur ayant pour argument le nom de la ville (en majuscules ou minuscules).
   - Un constructeur ayant pour argument le nom de la ville (en majuscules ou minuscules) ainsi que le nombre d'habitants.

   et les méthodes
   - Les accesseurs `getNom` et `getNbHabitants` qui renvoient respectivement le nom et le nombre d'habitants de la ville. Si le nombre d'habitants est inconnu, `getNbHabitants` renvoie 0.
   - Une méthode `setNbHabitants` qui permet de mettre à jour le nombre d'habitants. La nouvelle valeur est transmise en argument. Si elle est négative, la valeur de `nbHabitants` reste à 0.
   - Une méthode `nbHabitantsConnu` qui renvoie un booléen de valeur vrai si le nombre d'habitants est connu et faux sinon.
   - Une redéfinition de la méthode `toString`.

   On suppose que les données transmises en argument sont correctes. On ne demande pas de test et de gestion de situations d'erreur.
2. Définir la classe `Capitale`. Une capitale est une ville particulière. Elle a pour attribut supplémentaire le nom du pays dont elle est la capitale. Le nom du pays est en majuscules.
   - Définir les constructeurs
     `Capitale(String nom, String pays)` et
     `Capitale(String nom, String pays, int nbHabitants)`.
     Utiliser les mots clés `super` et `this`.
     Rappeler ce qui se produit si la première instruction d'un constructeur n'est pas `super()` ou `this()` (avec ou sans argument).
   - Redéfinir la méthode `toString`.
3. On est amené à classer les villes en 4 catégories :

   catégorie A : celles de moins de 500 000 habitants,
   catégorie B : celles de 500 000 habitants et plus,
   catégorie ? : celles dont le nombre d'habitants est inconnu,
   catégorie C : celles qui sont capitales d'un pays, quel que soit le nombre d'habitants.

   Dire comment et ou définir la méthode `categorie` qui renvoie l'un des 4 caractères `?`, `A`, `B` ou `C`. Utiliser les liaisons dynamiques. Votre code ne doit pas comporter de test sur le type des instances.