Programmation Orientée Objet
TP 4 : Héritage et composition.

https ://dpt-info.u-strasbg.fr/~kvanhoey

Introduction

Durant ce TP, nous allons voir ou revoir les notions associées à l'héritage : visibilité des attributs, affectation, liaison dynamique, super, redéfinition de méthodes.

1 Composition vs. héritage & attributs public/private/protected

On dispose de la classe suivante :

```
class Point
{
    public double x, y ;

    public Point(double x, double y)
    {
        this.x = x ;
        this.y = y ;
    }

    public void deplace(double dx, double dy)
    {
        x += dx ;
        y += dy ;
    }

    public String toString()
    {
        return "Point de coordonnees : " + x + ", " + y ;
    }
}
```

On souhaite réaliser une classe `Cercle` disposant des méthodes suivantes :
- constructeur recevant en argument les coordonnées du centre du cercle et son rayon ;
- *deplaceCentre* pour modifier les coordonnées du centre du cercle ;
- *changerRayon* pour modifier le rayon du cercle ;
- *getCentre* qui fournit en résultat un objet de type `Point` correspondant au centre du cercle ;
- *toString* qui renvoie sous forme de `String` les coordonnées du centre du cercle et de son rayon.

A) héritage et composition.

1. Définir la classe `Cercle` comme possédant un membre de type `Point` (composition).
2. Définir la classe `Cercle` comme classe dérivée de `Point` (mot clé **extends**).

Dans les deux cas, on écrira un petit programme mettant en jeu les différents fonctionnalités de la classe `Cercle` comme par exemple :

```
public class TestCercle
{
    public static void main(String[] args)
    {
        Cercle c = new Cercle(3, 8, 2.5) ;
        System.out.println(c) ;

        c.deplaceCentre(1, 0.5) ;
        c.changeRayon(5.25) ;
        System.out.println(c) ;

        Point pt = c.getCentre() ;
        System.out.println(pt) ;
    }
}
```

B) visibilité des attributs.

Dans la suite de l'exercice, la classe `Cercle` est dérivée de `Point`.

1. Modifier les deux classes afin que les attributs deviennent privés. Ajouter des accesseurs à la classe `Point` : *pubic double getX()* et *public double getY()*. Modifier la classe `Cercle` en conséquence.
2. Modifier la classe `Point` afin que ses attributs soient protected. Modifier la classe `Cercle` en conséquence. Tester l'accessibilité des attributs de `Point` dans le programme principal.

C) affectation, liaison dynamique.

La classe `Cercle` est dérivée de Point. Parmi les instructions suivantes, dire quelles sont acceptées, refusées. Expliquer pourquoi.

```
public class TestCercle
{
    public static void main(String[] args)
    {
        Point pt1, pt2 ;
        Cercle c1, c2 ;
        pt1 = new Point(4.2, 7.3) ;
        pt2 = new Cercle(14.7, 62, 5.2) ;
        c1 = new Cercle(3.56, 7.84, 2.25) ;
        c2 = new Point(56.1, 76.9) ;
        pt1.deplace(0.5, 0.5) ;
        pt2.deplace(0.3, 0.8) ;
        pt2.changerRayon(6.21) ;
        c1.deplace(0.4, 0.5) ;
        c1.changeRayon(2.52) ;
        c1.x = 4.78 ;
    }
}
```

2 super, redéfinition de méthode – 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 ("Lausanne") ;
        v2 = new Ville ("Strasbourg", 272975) ;
        System.out.println(v1) ;
        System.out.println(v2) ;
        System.out.println() ;

        Capitale c1, c2 ;
        c1 = new Capitale("Bruxelles", "Belgique") ;
        c2 = new Capitale("Luxembourg", "Luxembourg", 100390) ;
        c1.setNbHabitants(1300000) ;
        System.out.println(c1) ;
        System.out.println(c2) ;
        System.out.println() ;

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

qui lors de l'exécution affichera le résultat suivant :

```
Ville de Lausanne.
Ville de Strasbourg ; 272975 habitants.
Ville de Bruxelles ; 1300000 habitants. Capitale de Belgique.
Ville de Luxembourg ; 100390 habitants. Capitale de Luxembourg.

categorie de la ville de Lausanne : ?
categorie de la ville de Strasbourg : A
categorie de la ville de Bruxelles : C
```

Question : 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.
- 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).

et les méthodes :
- 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.
- 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 et redéfinir la méthode *toString*.

3. On est amené à classer les villes en 4 catégories :
   A : celles de moins de 500 000 habitants ;
   B : celles de 500 000 habitants et plus ;
   ? : celles dont le nombre d'habitants est inconnu ;
   C : celles qui sont capitales d'un pays, quel que soit le nombre d'habitants.
   Dire comment et où 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 variables.