

PROGRAMMATION PAR LES OBJETS EN JAVA

LES RELATIONS ENTRE CLASSES (TD4)

NAJIB TOUNSI

(Lien permanent: <http://www.mescours.ma/Java/TD/tdJava4.html> (.pdf))

L'objectif de ce TD est de créer une classe (Rectangle) *cliente* d'une autre classe (Point). Ensuite de dériver une classe (Robot) à partir d'une classe (Point). Notions d'héritage et de *polymorphisme*.

SOMMAIRE

1. [Relation d'utilisation](#)
2. [Relation d'héritage](#)
 1. [Essai simple héritage : rajout de champ et de méthode](#)
 2. [Essai simple héritage : polymorphisme](#)

1. RELATION D'UTILISATION

Reprendre la classe Point (avec constructeurs) du [TD précédent](#) et la compiler. Créer (nouveau fichier, Rectangle.java) une classe Rectangle, qui utilise cette classe Point.

```
1  class Rectangle {
2      private Point hg, bd ;      // Les coins haut à gauche
3                                  // Rectangle droit
4      public Rectangle(){
5          // rectangle par défaut. Choisir son initialisation
6      }
7
8      public Rectangle(Point h, Point b){
9          // initialisation des coins à partir des paramètres
10     }
11
12     public void afficher(){
13         // Affiche les coordonnées des coins
14     }
15
16     public int surface(){
17         // calcule de la surface
18     }
19
20     public void zoom(int deltax, int deltay){
21         // Dilatation des coordonnées. Delta donné.
22     }
23
24         // autres méthodes...
25     };
```

1. Programmer les méthodes et les tester. (Nouvelle classe `TestRectangle.java` avec méthode `main()`).
2. Rajouter à la classe `Point` un constructeur copie

```
Point (Point p) {
    // copier les coordonnées de p dans this ...
}
```

et l'utiliser dans le constructeur `Rectangle (Point h, Point b)`. Quel est l'intérêt par rapport à avant?

3. Rajouter les méthodes `set` et `get` appropriées qui modifient et lisent les champs d'un rectangle
4. Créer maintenant une classe `Fenetre`, composée d'un rectangle et d'un texte représentant le titre de la fenêtre.
Imaginer des opérations utiles, comme le déplacement, l'agrandissement d'une fenêtre, etc. Quelle opération de la classe `Rectangle` pourrait servir pour l'agrandissement d'une fenêtre?

2. RELATION D'HÉRITAGE

2.1. ESSAI SIMPLE HÉRITAGE : RAJOUT DE CHAMP ET DE MÉTHODE

1. Créer une sous-classe `Robot` de la classe `Point` et qui consiste en un point avec un champ `direction` (entier compris entre 1 et 4) et une méthode `avancer ()` qui fait avancer le robot dans un sens selon la valeur du champ `direction` (1 monter d'un pas, 2 à droite d'un pas, 3 descendre d'un pas et 4 à gauche d'un pas.)

```
1  class Robot extends Point {
2      int direction;          // direction actuelle
3
4      public Robot(){
5          super (0,0);
6          direction = 3;  // Sud
7      }
8      ...
9      public void avancer() {
10         switch (direction) {
11             case 1: y++; break;
12             case 2: x++; break;
13             case 3: y--; break;
14             case 4: x--; break;
15             default: ;
16         }
17     }
18 };
```

2. Faire un programme qui instancie un `Robot`, le fait avancer et imprime ensuite les coordonnées du `Robot`.
Utiliser `toString()` héritée de `Point` (elle même héritée de `public String toString()` de `Object`, et redéfinie dans `Point`).
3. Rajouter une méthode `setDirection (int d)` qui change la direction d'un `Robot` à la valeur `d`. Tester.
4. Rajouter un constructeur paramétré par trois entiers (les deux coordonnées et la direction) et qui initialise un robot avec ces valeurs.
Question : Peut-on dans ce cas supprimer le constructeur défaut `Robot()` ? Tester pour vérifier.
(Réponse : oui, à condition de ne pas l'appeler en faisant `new Robot()`, car sinon c'est le nouveau constructeur avec paramètres qui sera trouvé, et comme il ne correspond pas c'est une erreur de compilation. Dit autrement, le constructeur défaut hérité `Point()` est utilisé pour l'appel `new Robot()` si aucun constructeur n'est défini dans la sous-classe `Robot`)

2.2. ESSAI SIMPLE HÉRITAGE : POLYMORPHISME

1. Redéfinir pour un `Robot` la méthode `toString()` héritée de `Point`. Y rajouter aussi la direction en plus des coordonnées `x` et `y`.
2. Tester la suite d'instructions suivante :

```
Point p = new Point (6,7);  
System.out.println(p.toString());  
  
p = r;  
System.out.println(p.toString());
```

3. Constater le changement *dynamique* (à l'exécution) de méthode `toString()` appliquée à `p` dans les deux cas. D'abord celle de `Point` et ensuite celle de `Robot`, selon l'instance présente référencée par `p`.