Université de Béjaia
Faculté des Sciences Exactes
Département d'informatique

2ème Année Licence Informatique
Semestre : 04

Module : Programmation Orientée Objet

Cours élaboré par H. EL BOUHISSI

Héritage et Polymorphisme

Cours élaboré par H. EL BOUHISSI

Introduction

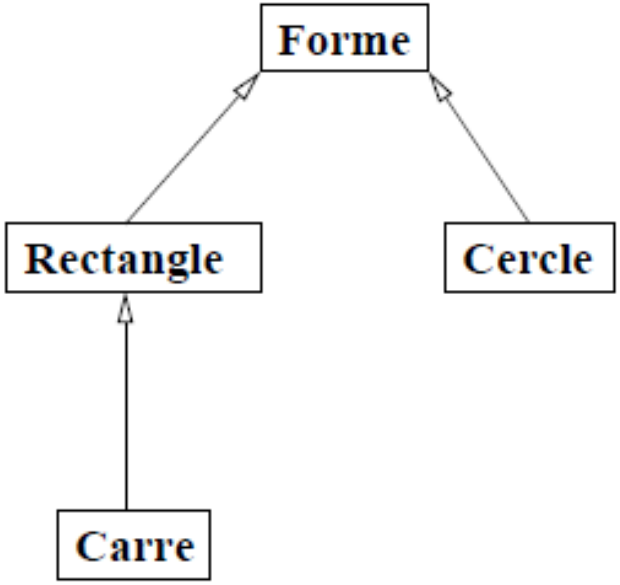
L'héritage, l'un des mécanismes les plus puissants de la programmation orientée objet, permet de reprendre des membres d'une classe (appelée superclasse ou classe mère) dans une autre classe (appelée sous-classe, classe fille ou encore classe dérivée), qui en hérite. De cette façon, on peut par exemple construire une classe par héritage successif. En Java, ce mécanisme est mis en œuvre au moyen du mot-clé **extends**

L'héritage est un mécanisme qui permet de réutiliser le même code ainsi que sa gestion selon les besoins. Il permet aussi de définir une hiérarchie descendante entre les classes, qui sont:

- La super-classe ou classe mère qui est la classe de base
- La ou les classes filles qui héritent de cette classe mère

Le mécanisme d'héritage

L'idée principale de l'héritage est d'organiser les classes de manière hiérarchique. La relation d'héritage est unidirectionnelle et, si une classe B hérite d'une classe A, on dira que B est une sous-classe de A. Cette notion de sous-classe signifie que la classe B est un cas particulier de la classe A et donc que les objets instanciant la classe B instancient également la classe A.



Prenons comme exemple des classes **Carre**, **Rectangle** et **Cercle**. La figure ci-contre propose une organisation hiérarchique de ces classes telle que **Carre** hérite de **Rectangle** qui hérite, ainsi que **Cercle**, d'une classe **Forme**.

Exemple

```
public class Vehicule
{
    public int vitesse;
    public int nombre_de_places;
}

public class Automobile extends Vehicule
{
    public Automobile()
    {
        this.vitesse = 90;
        this.nombre_de_places = 5;
    }
}
```

Dans cet exemple, la classe ***Automobile*** hérite de la classe ***Vehicule***, ce qui veut dire que les attributs ***vitesse*** et ***nombre_de_places***, bien qu'étant définis dans la classe ***Vehicule***, sont présents dans la classe ***Automobile***.

Le constructeur défini dans la classe Automobile permet d'ailleurs d'initialiser ces attributs.

Exercice

Soit le programme suivant se trouvant dans une classe quelconque: (Les notations ne correspondent pas à celles de java.

Soit une106, une instance de Voiture et monVehicule, une instance de Véhicule.

classe Vehicule	classe Voiture dérive de Véhicule{
méthode rouler(){	méthode seGarer(){
... }	...
méthode seGarer(){	}
... }	méthode klaxonner(){
	...
	}
	}

1. Peux-t-on faire une 106.rouler() ? Pourquoi ?
2. Si l'on fait une 106.seGarer(), est-ce la méthode de la classe Véhicule ou de la classe Voiture qui est appelée ?
3. Peux-t-on faire monVehicule.klaxonner() ? Pourquoi ?

Solution

1. Oui car une106 est un objet de type Voiture qui dérive de Véhicule.
2. C'est la méthode de Voiture car par défaut, la méthode exécutée est celle de la classe courante.
3. Non car la classe Véhicule ne possède pas de méthode klaxonner().

(Redenition)
Les exemples suivants sont-ils corrects ?
Justiez.

1.
```
class A {
    public void f() { System.out.println("Bonjour."); }
}
class B extends A {
    private void f() { System.out.println("Bonjour les amis."); }
}
```
Correction : Non : on ne peut redéfinir une méthode en restreignant sa visibilité.

2.
```
class A {
    public int f(int a) { return a++; }
}
class B extends A {
    public boolean f(int a) { return (a==0);}
}
```
Correction : Non : on ne peut redéfinir une méthode et changer le type du resultat.

3.
```
class A {
    public int f(int a) { return a++; }
}
class B extends A {
    public int f(int a, int b) { return a+b;}
}
class test {
    B obj = new B();
    int x = obj.f(3);
    int y = obj.f(3,3);
}
```
Correction : Oui. L'appel `f(3)` utilise la méthode de la super classe.

4.
```
class A {
    public int f(int a) { return a++; }
}
class B extends A {
    public int f(int a, int b) { return a+b;}
}
class test {
    A obj = new B();
    int x = obj.f(3);
    int y = obj.f(3,3);
}
```

Correction : Non, le complilateur va réjecter le programme car `obj` a été declaré de la classe `A` pour le quel `f` est défini sur un seul argument.