Héritage

1. Pourquoi

① L'héritage existe dans le monde réel.

② Parmi les objectifs de la POO : s'approcher du monde réel.

① et ② ⇨ l'héritage doit exister dans les concepts objet.

Exemple

- Un Rectangle est un Polygone à quatre cotés, tous ses angles sont droits,
- Un Carre est un Polygone à quatre cotés, tous ses angles sont droits et sa longueur est égale à sa largeur

⇨ Un Carre est un Rectangle sa longueur est égale à sa largeur

C'est une définition plus simple qui utilise l'héritage, d'ailleurs dans la définition du Rectangle on a utilisé l'héritage « est un Polygone »

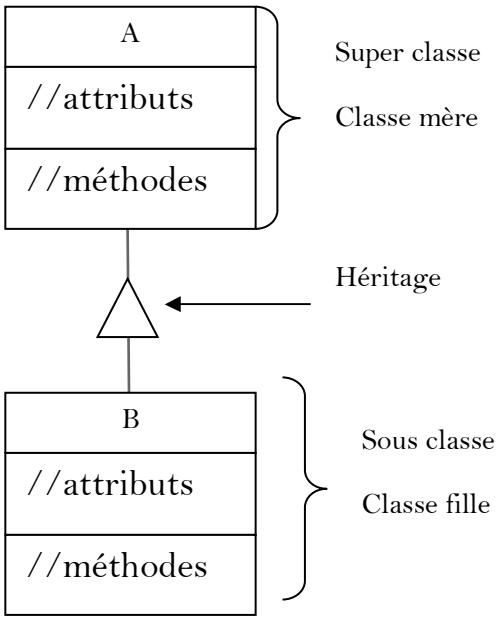
2. Définition

L'héritage est une relation entre deux classes de type « est un » (is a).

Exemple

- Cercle : pas d'héritage
- Carre : pas d'héritage
- Ellipse, Cercle : il ya un héritage : un Cercle est une Ellipse
- Rectangle, Carre : il ya un héritage : un Carre est une Rectangle

3. Représentation UML



4. Syntaxe

```
public class SousClasse extends SuperClasse{
}
```

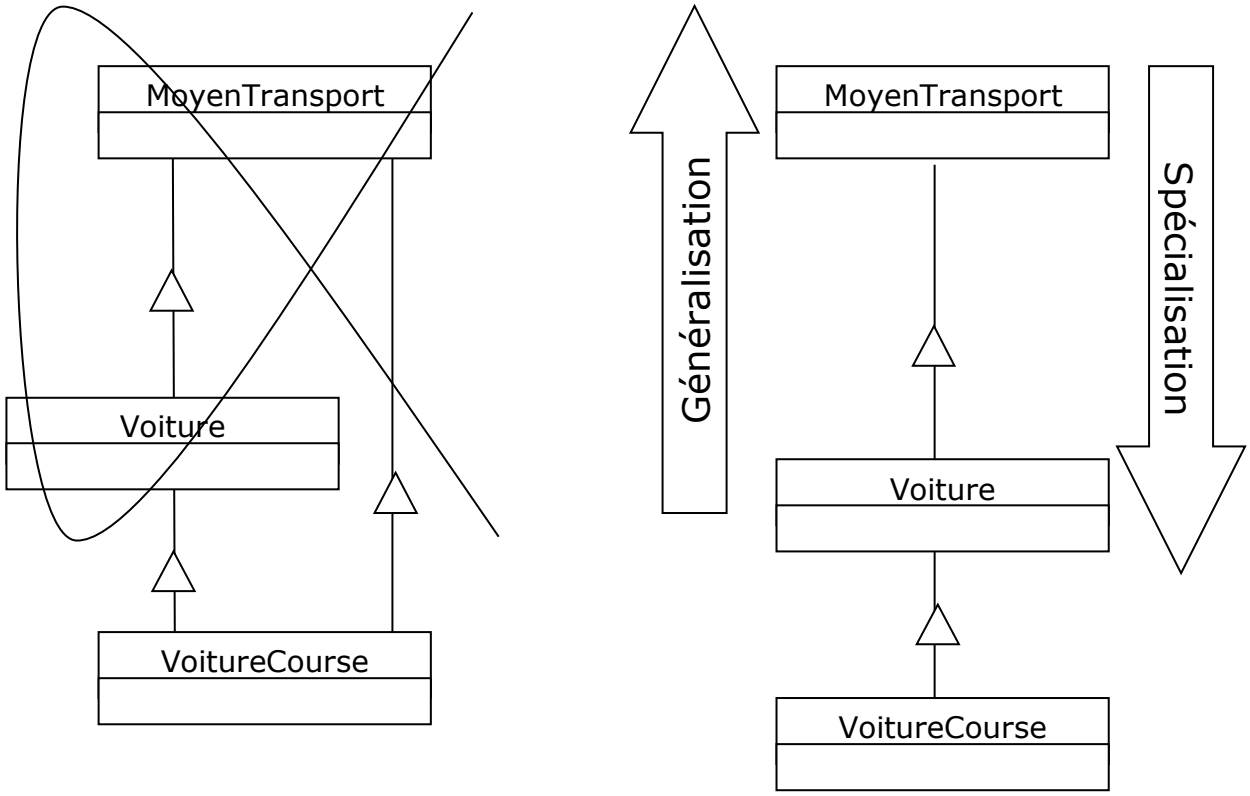
Exemple

```
public class Rectangle extends Polygone{
}
public class Carre extends Rectangle{
}
```

Remarque

MoyenTransport, Voiture, VoitureCourse:

- Une Voiture est un MoyenTransport
- Une VoitureCourse est un MoyenTransport
- Une VoitureCourse est une Voitures



5. Qu'est-ce que la sous classe hérite

5-1 Attributs

La sous classe hérite les attributs public et protected.

Exemple

```
public class A{
        public int i ;
        protected int j ;
        private int k ;
}
public class B extends A{
        public void f(){
                i=5 ;
                j=5 ;
                k=5 ;
        }
}
//Dans le main
A a = new A() ;
a.i=5 ;
a.j=5 ;
a.k=5 ;
B b = new B() ;
b.i=5 ;
b.j=5 ;
b.k=5 ;
```

5-2- Constructeurs

La sous classe hérite seulement le constructeur sans paramètres (par défaut) et ceci seulement si :

- ce constructeur est défini explicitement dans la super classe et sa visibilité est public ou protected
- ou
- la super classe ne définit aucun constructeur explicite

Exemple

```
public class A{
        private int i ;
        public A(){
                i=0 ;
        }
        public A(int i){
                this.i=i ;
        }
}
public class B extends A{
}
```

Le seul constructeur qui sera hérité par B de A sera le constructeur sans paramètres, son nom dans B devient B().

```
B b1 = new A() ;

B b2 = new B() ;
```

5-3- Méthodes

A- Méthodes héritées

La sous classe hérite les méthodes public et protected.

Exemple

```
public class A{
        public void f(){
        }
        protected void g(){
        }
        private void h(){
        }

}
public class B extends A{
        public void k(){
                f();
                g() ;
                h() ;
        }
}
```

B- Redéfinition d'une méthode

On dit qu'on a une redéfinition de méthode si :

- On a une super classe et sa sous classe
- La super classe et sa sous classe possèdent une méthode :
  - de même nom,
  - de même signature,
  - cette méthode est public ou protected dans la super classe (c.à.d. qu'elle est héritée par la sous classe).

Lorsqu'une instance de la sous classe appelle la méthode redéfinie, c'est la méthode redéfinie par la sous classe qui sera exécutée.

<table>
<tr>
<td>

```
public class A{
        public void f(){
        }
}
public class B{
        public void f(){
        }
}
```

Pas de Redéfinition

</td>
<td>

```
public class A{
        public void f(){
        }
}
public class B extends A{
        public void f(int i){
        }
}
```

Pas de Redéfinition

</td>
</tr>
<tr>
<td>

```
public class A{
        private void f(){
        }
}
public class B extends A{
        public void f(){
        }
}
```

Pas de Redéfinition

</td>
<td>

```
public class A{
        public void f(){
        }
}
public class B extends A{
        public void f(){
        }
}
```

Redéfinition

</td>
</tr>
</table>

On utilise une redéfinition (@Override)de méthode lorsque:

- La méthode dans la sous classe possède un traitement supplémentaire à celui dans la méthode de la super classe (exemple afficher dans Employe affiche le salaire en plus des données affichées par afficher de Personne).
- La méthode dans la sous classe possède une optimisation de calcul (exemple getPerimetre dans Carre utilise la formule 4*longueur (une seule multiplication) par contre dans Rectangle elle utilise la formule 2*(longueur+largeur) (une multiplication et une addition)
- La méthode dans la sous classe possède un traitement différent à celui de la méthode dans la super classe (equals plus tard).

6. Le mot clé super

6-1- Appel d'un constructeur de la super classe

```
public class Rectangle{
        protected float langueur;
        protected float largeur;
        public Rectangle(float longueur, float largeur){
                this.longueur = longueur ;
                this.largeur = largeur ;
        }
```

```
        public Rectangle(){
                this(1,1) ;
        }
}
public class Carre extends Rectangle{
        public Carre(float cote){ // Premier constructeur
                // Solution1
                super();
                longueur=cote;
                largeur = cote;
        }

        //Deuxième constructeur
        //Solution1
        //ne rien écrire le constructeur sans paramètre sera hérité

        //Solution2
        public Carre(){
                //super() ; sera appelé implicitement
        }
        //Solution3
        public Carre(){
                super() ;  //appel explicite du constructeur sans paramètres de Rectangle
        }
        //Solution4
        public Carre(){
              super(1,1) ; //appel explicite du constructeur avec paramètres de Rectangle
        }
        //Solution
        public Carre(){
                this(1) ;  //appel explicite du constructeur avec paramètres de Carre
        }
}
```

// Solution2
//super(); est appelé implicitement
longueur=cote;
largeur = cote;

// Solution3
super(cote, cote);

6-2- Accès à un attribut de la super classe

```
public class A{
        public int i ;
        protected int j ;
        private int k ;
        public A(){
                i=10 ;
                j=20 ;
                k=30 ;
        }
}
public class B extends A{
        private int i ;
        private  int j ;
        private int k ;
```

```
        public B(){
                i=10 ;
                j=20 ;
                k=30 ;
                super.i=40;
                super.j=50;
                super.k=60;
        }
}
```

6-3- Appel d'une méthode de la super classe

```
//1er cas
public class A{
        public void f(){
                sysout("A");
        }
}
public class B extends A{
}
A a=new A() ;
a.f() ;                 //correct et affiche A
B b=new B() ;
b.f();                  //correct et affiche A

//2ème  cas
public class A{
        public void f(){
                sysout("A");
        }
}
public class B extends A{
        public void f(){
                sysout("B");
        }

}
A a=new A() ;
a.f() ;                 //correct et affiche A

B b=new B() ;
b.f();                  //correct et affiche B
//3ème  cas
public class A{
        public void f(){
                sysout("A");
        }
}
public class B extends A{
        public void f(){
                f();                    //provoque une boucle infini
                sysout("B");
        }
}
```

```
A a=new A() ;
a.f() ;                    //correct et affiche A
B b=new B() ;
b.f();                     //correct  n'affiche rien et entre dans la boucle infinie

//5ème  cas
public class A{
        public void f(){
                sysout("A");
        }
}
public class B extends A{
        public void f(){
                super.f() ;
                sysout("B");
        }
}
A a=new A() ;
a.f() ;                    //correct et affiche A
B b=new B() ;
b.f();                     //correct  affiche AB

//6ème  cas
public class A{
        public void f(){
                sysout("A");
        }
}
public class B extends A{
        public void f(){
                sysout("B");
                super.f() ;
        }
}
A a=new A() ;
a.f() ;                    //correct et affiche A
B b=new B() ;
b.f();                     //correct  affiche BA
```

7.7- La classe Object

La classe Object est la super classe de toutes les classes Java. Toute classe est une sous classe de la classe Object d'une façon directe ou indirecte. C'est la seule classe qui connait et effectue la réservation de mémoire pour les attributs.

public class A{} est équivalent à public class A extends Object{}

7-1- La méthode toString()

```
public String toString(){
        //Retourne une description textuelle de la classe
}
```

Dans la classe Object cette méthode retourne la concaténation de :

- nom du package,
- nom de la classe,
- la référence mémoire de l'objet.

Si la sous classe ne redéfinie pas cette méthode alors elle obtient ce résultat.

Exemple :

```
//1er cas la classe Stylo ne redéfinit pas la méthode toString
Stylo s=new Stylo() ;
sysout(s.toString) ;   //ou aussi sysout(s);
Ceci donne à l'exécution : stylo.Stylo @FF44
//2ème cas la classe Stylo redéfinit la méthode toString
public class Stylo{
        …
        public String toString(){
                return marque+ "  "+couleur ;
        }
}
Stylo s=new Stylo() ;
sysout(s.toString) ;   //ou aussi sysout(s);
```

Ceci donne à l'exécution : BIC Bleu

7-2- La méthode equals()

```
public boolean equals(Object arg0){
}
```

```
== permet de déclarer les références (@ mémoire)
Stylo s1=new Stylo() ;
Stylo s2=new Stylo() ;
Stylo s3=s1 ;
if(s2==s1)
        sysout("s1=s2");
else
        sysout("s1!=s2");
```

Bien que s1 et s2 ont tous les attributs identique ce code affiche s1!=s2 car @s1 !=@s2

```
if(s3==s1)
        sysout("s1=s3");
else
        sysout("s1!=s3");
```

Ce code affiche s1!=s3 car @s1 !=@s3

La méthode equals dans la classe Object possède le même fonctionnement que ==, si une sous classe possède une autre signification de equals elle doit la redéfinir

8. L'opérateur instanceof

L'opérateur instanceof permet de tester si un objet est une instance (à pour type) une classe.

Syntaxe :

objet instancefo Classe //ceci retourne un boolean

Exemple :

```
Polygone p=new Polygone() ;
Rectangle r= new Rectangle() ;
Carre c=new Carre() ;
p instanceof Object          //donne true
p instanceof Polygone        //donne true
p instanceof Rectangle       //donne false
p instanceof Carre           //donne false
r instanceof Object          //donne true
r instanceof Polygone        //donne true
r instanceof Rectangle       //donne true
r instanceof Carre           //donne false
c instanceof Object          //donne true
c instanceof Polygone        //donne true
c instanceof Rectangle       //donne true
c instanceof Carre           //donne true
```

Deux stylo sont égaux si il ont la même marque et la même couleur, pour obtenir ceci il faut redéfinir equals.

```
public class Stylo{
        …
        public boolean equals(Object arg0){
                if(arg0 instanceof Stylo){
                        Stylo s=(Stylo)arg0; //conversion de type force
                                             // ou transtypage ou cast
                        If((marque.equals(s.marque)
                          &&(couleur.equals(s.couleur) )
                                return true ;
                }
                return false ; //un Stylo ne peut pas être égale à quelque chose qui n'est pas un stylo
        }
}
```

La classe String a redéfinie la méthode equals pour qu'elle compare les caractères et non pas les références.
Dans ce cas

```
Stylo s1=new Stylo() ;
Stylo s2=new Stylo() ;
Stylo s3=s1 ;
if(s2==s1)
        sysout("s1=s2");
else
        sysout("s1!=s2");
```

Ce code affiche s1=s2

9. Bloquer l'héritage

Pour bloque l'héritage à partir d'une classe, on utilize le mot clé final devant la classe.

```
public final class Carre extends Rectangle{

}
Public CarreSpecial extends Carre{

}
```

10. Interdire la redéfinition

Pour interdire la redéfinition d'une méthode on utilise le mot clé final devant la méthode.

```
public class Carre extends Rectangle{

    public final float getPerimetre(){

        return 4*longueur;

    }

}

Public CarreSpecial extends Carre{

    public float getPerimetre(){

        return 4*longueur;

    }

}
```

11. Héritage simple et héritage multiple

Java ne supporte que l'héritage simple. Après extends il n'y a qu'une seule classe.