

# Polymorphisme

## 1. PLAN

Quoi  
Surcharge  
Redéfinition  
Polymorphisme de référence  
Liaison dynamique  
Transtypage ou conversion de type forcée (cast)  
Structure polymorphe

### 1- Quoi

Poly : plusieurs

Morphisme : forme

Donc le polymorphisme concerne les choses qui peuvent avoir plusieurs formes :

- des méthodes de même nom qui ont plusieurs comportements
- une référence qui contient des objets de types différents
- un tableau qui contient des objets de types différents

### 2- SURCHARGE

C'est une forme de polymorphisme, en effet on appelle les méthodes surchargées avec le même nom et le comportement change suivant les paramètres de l'appel.

```
public class A {  
    public void f() {  
        System.out.println("f de A");  
    }  
    public void f(int i) {  
        System.out.println("f"+i+ "de A");  
    }  
}  
A a = new A();  
a.f();          ---- >      f de A  
a.f(1);         ---- >      f1 de A
```

### 3- REDEFINITION

C'est une forme de polymorphisme, en effet on appelle les méthodes redéfinies avec le même nom et le comportement change suivant l'instance qui fait l'appel.

```
public class A {  
    public void f() {  
        System.out.println("f de A");  
    }  
}  
public class B extends A {  
    public void f() {  
        System.out.println("f de B");  
    }  
}  
A a = new A();  
a.f();          ---- >      f de A  
B b = new B();  
b.f();          ---- >      f de B
```



## 4- POLYMORPHISME DE REFERENCE

```
public class A {
    public void f() {
        System.out.println("f de A");
    }
}

public class B extends A{
    public void f() {
        System.out.println("f de B");
    }
    public void g() {
        System.out.println("g de B");
    }
}

A a;
a = new A();           //correct a est de type A
                        //et A possède un constructeur sans paramètres
a.f();                 //correct a est de type A
                        //et A possède la méthode f

a = new B();           //correct a est de type A
                        //et B est une sous classe de A
                        //et B possède un constructeur sans paramètres
```

a est de type A, en premier lieu elle a contenue une instance de la classe A, en second lieu elle a contenue une instance de la classe B : c'est le polymorphisme de référence.

a ne peut invoquer que les méthodes de la classe de déclaration(A).

```
a.f();                 //correct a est de type A
                        //et A possède la méthode f
a.g();                 //erreur de compilation : a est de type A
                        //et A ne possède pas la méthode g
```

### REMARQUE

```
B b= new A();          // erreur de compilation
                        //la classe A n'est pas une sous classe de B
```

## 5- LIAISON DYNAMIQUE

Lorsqu'une instance appelle une méthode, c'est la méthode de l'objet chargée en mémoire qui est exécutée et pas nécessairement celle du type de déclaration.

### - 1<sup>er</sup> cas

```
public class A {
    public void f() {
        System.out.println("f de A");
    }
}

public class B extends A{
    public void f() {
        System.out.println("f de B");
    }
}

A a;
a = new A();
a.f();                 //affiche f de A
a=new B();
a.f();                 //affiche f de B
```



### - 2<sup>ème</sup> cas

```
public class A {
    public void f() {
        System.out.println("f de A");
    }
}

public class B extends A{
}

A a;
a = new A();
a.f();           //affiche f de A
a=new B();
a.f();           //affiche f de A
```

### - 3<sup>ème</sup> cas

```
public class A {
}

public class B extends A{
    public void f() {
        System.out.println("f de B");
    }
}

A a;
a = new A();
a.f();           //erreur de compilation
a=new B();
a.f();           //erreur de compilation
```

### - 4<sup>ème</sup> cas

```
public class A {
}

public class B extends A{
}

A a;
a = new A();
a.f();           //erreur de compilation
a=new B();
a.f();           //erreur de compilation
```

## 6- TRANSTYPAGE OU CONVERSION DE TYPE FORCEE (CAST)

```
public class A {
    public void f() {
        System.out.println("f de A");
    }
}

public class B extends A{
    public void f() {
        System.out.println("f de B");
    }

    public void g() {
        System.out.println("g de B");
    }
}

A a;
a=new B();
a.g();           //erruer de compilation
```

Bien que l'objet chargé dans la référence a est un B, on ne peut pas appelé la méthode g() de B parce que a est déclaré comme étant un A. Pour pouvoir appeler la méthode g il faut faire un transtypage.

```
Solution 1 :    ((B)a).g();           //correct et affiche g de B
Solution 2 :    B b = (B)a;
                b.g();                 //correct et affiche g de B
```



## Remarque

Même si la conversion de type est acceptée à la compilation, si à l'exécution cette conversion n'est pas possible, une erreur d'exécution est alors générée.

```
Polygone p1;  
p1=new Carre();           //correct  
Rectangle r=(Rectangle)p1; //correct  
                        //p1 est un carre il peut être converti en Rectangle  
Carre c = (Carre)p1;      //correct  
                        //p1 est un carre il peut être converti en Carre  
  
Polygone p2;  
p2=new Rectangle();       //correct  
Rectangle r=(Rectangle)p2; //correct  
                        //p1 est un rectangle il peut être converti en Rectangle  
Carre c = (Carre)p2;      //erreur d'exécution  
                        //p1 est un Rectangle il ne peut pas être converti en Carre
```



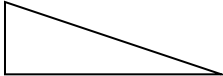

Pour faire face à ce problème, généralement la conversion de type est précédé par un test de type instanceof

```
if(p instanceof Carre){  
    Carre c =(Carre)p;  
    System.out.println(p.getSurface);  
}
```

## 7- STRUCTURE POLYMORPHE

Ce ci consiste à déclarer un structure de type tableau qui contient des instances de classe différentes.

```
Polygone [] tPolygone;  
tPolygone = new Polygone[4];  
tPolygone[0]=new Rectangle(6,3);  
tPolygone[1]=new Carre(3);  
tPolygone[2]=new Triangle();  
tPolygone[3]=new Rectangle(6,3);
```

			
0	1	2	3

Les méthodes visibles de chaque case du tableau sont ceux de la classe qui a servi à la déclaration.

Si on généralise la classe de déclaration du tableau alors on augmente les instances que peut contenir le tableau mais on diminue les méthodes visibles pour chaque case du tableau.

Si le tableau est déclaré de type Object alors il peut contenir n'importe quelle instance cependant les méthodes visibles sont seulement ceux de la classe Object.

## Règle

Dans le choix du type du tableau on choisit la classe générique la plus basse à partir de laquelle dérive toutes les classes des instances du tableau.

Si le tableau contient des Rectangle et des carre il peut être déclaré tableau de Object ou de Polygone ou de Rectangle on choisit de le déclarer comme tableau de Rectangle parce que c'est la classe générique la plus basse.

