



# Introduction aux génériques

CREATION : 2010/06/17

MISE A JOUR : 2010/10/16

Les génériques ont été introduits par la plateforme 1.5 sous la pression des développeurs qui voulaient quelque chose d'aussi efficace que ce qui existait en C++.

Dans la suite, on suppose que la classe Point dispose d'une méthode getX()

## Legacy

En version 1.4 et inférieure, voilà ce que l'on pouvait écrire. On appelle cela le java historique ou Legacy

```
class Historique {
    Point p ;
    Object o ;
}
```

Dans l'objet o, on peut mettre ce que l'on veut, pourvu que ce soit un objet (Object est la classe mère de tous les objets) :

```
o =3 // NON
o = new int ; // NON PLUS
o = new Integer() ; // OK
o = new Point(); //OK
```

Les ennuis commencent selon l'usage de o : qu'est-ce que o ? Peut-il y avoir un problème ?

```
o.toString() // PAS DE PROBLEME
o.getX() // ERREUR A LA COMPILATION : getX() n'est pas dans l'interface d'Object
```

Il est nécessaire de faire un downcasting :

```
((Point) o).getX() ; // OK
((Integer)o).getX(); // ERREUR, à la compilation
```

Le problème vient de la vérification du type de o qui se fait à l'exécution. Si o n'est pas du bon type, le downcasting n'est pas possible et une exécution est levée. Seule manière d'être sur, c'est de vérifier le type de l'objet à l'exécution (ou de gérer l'exception avec un bloc try/catch) mais c'est lourd et coûteux.

```
If (o instanceof Point) ((Point)o).getX() ;
```



## Classe paramétrée

Une solution élégante à ce problème est d'utiliser les génériques introduits en version 1.5. Une classe peut être paramétrée par une autre classe que l'on donne à l'instanciation de la classe paramétrée. Le paramètre est en général (par convention) une simple lettre. La classe paramétrée est également compilée (différent du C++)

```
class Parametree<O> {
    Point p ;

    O o ;
}
```

La classe est instanciable de la manière suivante :

```
Parametree<Integer> p1 = new Parametree<Integer>(); // OK
Parametree<int> p2 = new Parametree<int>(); // NE MARCHE TOUJOURS PAS
Parametree<Point> p3 = new Parametree<Point>(); // OK
```

Si O représente la classe Point, il est impossible d'écrire o = new Integer(); on aura une erreur de compilation. O.getX() pourra être écrit directement, sans downcasting, ce qui sécurise bien notre application.

## Compléments

Il est possible de poser des conditions sur les paramètres de classe. Par exemple, la classe O doit étendre telle classe ou implémenter telle interface :

```
<O extends Classe>
<O extends Interface>
```

Il est possible d'écrire des méthodes paramétrées. Il existe également des wildcards (comme ?) pour préciser quel type de classe est utilisable.

Il n'est pas possible de créer un nouvel objet de la classe donnée en paramètre.

```
O o = new O(); // ne marchera pas.
```

Ce problème est illustré en TP.

Pour une utilisation un peu plus avancée, je vous conseille le tutoriel : (qui n'en n'est pas un ;-)) <http://www.oracle.com/technetwork/java/javase/generics-tutorial-159168.pdf>