

# Les mots réservés du langage Java

par Baptiste Wicht ([home](#))

Date de publication : 9 Janvier 2007

Dernière mise à jour :

Cet article va vous présenter une liste des tous les mots réservés du langage Java et leur signification.

A - Introduction

B - Les mots réservés

C - Significations

abstract

assert

boolean

break

byte

case

catch

char

class

const

continue

default

do

double

else

enum

extends

false

final

finally

float

for

goto

if

implements

import

instanceof

int

interface

long

native

new

null

package

private

protected

public

return

short

static

strictfp

super

switch

synchronized

this

throw

throws

transient

true

try

void

volatile

while

IV - Remerciements

## A - Introduction

Tout d'abord, qu'est ce qu'un mot réservé ? C'est tout simplement un mot clé du langage, par exemple true.

Qu'est ce que cela implique pour le développeur ? Il ne peut pas employer un de ces mots comme identifiant pour une variable, un nom de classe, un nom de package ou de méthode. C'est à dire qu'on ne pourra pas avoir des choses telles que celles-ci :

```
Boolean true = new Boolean(true);  
public class true{}  
package com.wichtounet.true;
```

Tous ces exemples, vont lever une erreur. Par contre, vous pouvez tout à fait les employer à l'intérieur d'un mot ou en en changeant la casse:

```
Boolean trueBoolean = new Boolean(true);  
public class TrueBoolean{}  
package com.wichtounet.trueBoolean;
```

## B - Les mots réservés

Voici une liste des mots réservés du langage Java :

- **abstract**
- **assert**
- **boolean**
- **break**
- **byte**
- **case**
- **catch**
- **char**
- **class**
- **const**
- **continue**
- **default**
- **do**
- **double**
- **else**
- **extends**
- **false**
- **final**
- **finally**
- **float**
- **for**
- **goto**
- **if**
- **implements**
- **import**
- **instanceof**
- **int**
- **interface**
- **long**
- **native**
- **new**
- **null**
- **package**
- **private**
- **protected**
- **public**
- **return**

- **short**
- **static**
- **strictfp**
- **super**
- **switch**
- **synchronized**
- **this**
- **throw**
- **throws**
- **transient**
- **true**
- **try**
- **void**
- **volatile**
- **while**

## C - Significations

Voici les différentes significations pour tous les mots clés

### abstract

Modificateur utilisé dans la déclaration d'une classe ou d'une méthode. Il indique que cette méthode ou cette classe est *abstraite*.

Une classe doit être déclarée abstraite dès le moment où une de ses méthodes est abstraite. Une classe abstraite ne peut pas être instanciée, il faut automatiquement créer une autre classe qui l'étend. Une classe abstraite peut contenir des méthodes abstraites et des méthodes normales.

Une méthode abstraite n'a pas de corps et doit obligatoirement être redéfini par la classe étendant la classe ou se trouve cette méthode. Une méthode abstraite ne pourra être déclarée ni static, ni final, ni private.

A noter qu'une interface est implicitement abstraite, ainsi que toutes ses méthodes.

Voici un petit exemple de classe abstraite :

```
public abstract class Personne{
    public abstract String getNom(){}

    public String toString(){
        return "Mon nom est " + getNom();
    }
}
```

### assert

Ce mot-clé permet de s'assurer de certaines choses avant de continuer l'exécution. C'est la programmation par contrat. On vérifie souvent les paramètres d'une méthode avec des assert pour être sûr qu'ils sont tels qu'on les pense.

Une AssertionError est lancée si la condition n'est pas vérifiée. Les assertions sont ignoré par défaut (il faut utiliser l'option -ea ou -enableassertions pour les activer).

```
private double divise(int a, int b){
    assert b != 0 : "Impossible de diviser par zéro";
}
```

### Plus d'infos.

Ce mot-clé a été rajouté dans la version 1.4 de Java.

### boolean

boolean est un type primitif de Java. C'est un type indiquant si quelque chose est vrai ou faux. Il peut avoir comme valeur true ou false. Ce mot peut désigner un type de variable, de retour d'une méthode ou un paramètre.

```
boolean condition = true;
```

## break

Ce mot est une instruction permettant de sortir d'une instruction de contrôle, d'un opérateur d'itération ou d'un bloc try.

```
for(int i = 0; i < 1555; i++){  
    if(condition){  
        break; //On sort de la boucle  
    }  
}
```

Plus d'infos.

## byte

byte est un type primitif de Java. Il peut avoir comme valeur un entier signé représenté par 8 chiffres binaires. Ce mot peut désigner un type de variable, de retour d'une méthode ou un paramètre.

```
byte variable = 1;
```

## case

Ce mot précède une étiquette de l'instruction de contrôle switch.

```
switch(id) {  
    case 1 : System.out.println("Je suis premier"); break;  
    case 2 : System.out.println("Je suis deuxième"); break;  
    case 3 : System.out.println("Je suis troisième"); break;  
    default : System.out.println("Je ne sais pas où je suis");  
}
```

## catch

Clause servant à attraper une exception et à la traiter. Un bloc catch (clause catch plus le code la suivant) suit toujours un bloc try. Si une exception est levée à l'intérieur du bloc try, on arrive dans le bloc catch et on peut traiter l'exception.

```
try{  
    //Quelque chose  
}catch(UneException e){  
    //Si UneException est levée, on arrive ici  
}
```

## char

char est un type primitif de Java. Il représente un caractère. Ce mot peut désigner un type de variable, de retour d'une méthode ou un paramètre.

```
char variable = 'a';
```

## class

Ce mot définit une classe. Toute déclaration de classe doit commencer ainsi.

```
public class Test {  
    //...  
}
```

## const

Ce mot-clé n'est pas utilisé actuellement.

## continue

Ce mot permet de sauter une itération d'une boucle. C'est à dire qu'on passe directement à l'itération suivante sans effectuer le reste des opérations.

```
while(i > 100){  
    if(i % 25 == 0){  
        continue; //On passe à l'itération suivante, on ne va donc pas effectuer la suite  
    }  
  
    //Autres opérations  
}
```

## Plus d'infos.

## default

Ce mot représente l'étiquette de l'instruction dont le code sera effectué si aucune des autres conditions n'est remplie.

```
switch(i){  
    case 1 :  
        System.out.println("i est égal à 1");  
        break;  
    case 2 :  
        System.out.println("i est égal à 2");  
        break;  
    default :  
        System.out.println("i n'est pas égal à 1 ou 2");  
}
```

## do

Ce mot introduit une boucle do... while (fait... tant que), c'est une boucle dont la condition d'itération est vérifiée après l'exécution du corps de l'itération.

```
do{  
    //Opérations  
} while(condition)
```

## double

double est un type primitif de Java. Il peut avoir comme valeur un réel signé représenté par 64 chiffres binaires. Ce mot peut désigner un type de variable, de retour d'une méthode ou un paramètre.

```
double variable = '2.25';
```

## else

Ce mot introduit la partie (facultative) de l'opérateur de condition if qui est exécutée si la condition if n'est pas remplie.-

```
if(i == 2){
    System.out.println("i est égal à 2");
}else{
    System.out.println("i n'est pas égal à 2");
}
```

## enum

Ce mot-clé déclare une énumération, c'est à dire un type qui n'accepte qu'un ensemble fini d'éléments.

```
public enum Season {
    spring,summer, autumn, winter;
}
```

### Plus d'infos. Article sur les nouveautés de Java 5.0 : Les enum

Ce mot-clé a été rajouté dans la version 5.0 de Java.

## extends

Ce mot est utilisé dans la définition d'une classe ou d'une interface pour faire hériter une classe d'une autre. C'est à dire que la première classe va hériter des propriétés de la deuxième classe. extends est toujours suivi d'une classe.

```
public class Acteur extends Personne{ //Acteur hérite de Personne
    //...
}
```

## false

Ce mot représente une valeur du type boolean, elle indique une négation.

```
boolean variable = false;
```

## final

Ce mot-clé est un modificateur qui peut s'appliquer à une méthode, une variable ou une classe :

- Dans la déclaration d'une classe, final indique que celle-ci ne pourra pas être étendue
- Dans la déclaration d'une méthode, final indique qu'elle ne peut pas être redéfinie (si la méthode est static ou private, elle est automatiquement aussi final).
- Dans la déclaration d'une variable, final indique que cette variable est constante, on ne pourra donc l'affecter qu'une seule fois. Cela va aussi nous permettre d'utiliser la variable dans une classe anonyme.

### Plus d'infos.

## finally

Clause suivie d'un bloc d'instructions. Cette clause suit elle-même un bloc try, ou un bloc try-catch. Quelque soit la manière dont on sort du try, les instructions contenues dans le finally sont toujours exécutées.

```
try {  
    //Instructions quelconques  
} finally {  
    //Opérations effectuées après les instructions du try  
}
```

## float

float est un type primitif de Java. Il peut avoir comme valeur un réel signé représenté par 32 chiffres binaires. Ce mot peut désigner un type de variable, de retour d'une méthode ou un paramètre.

```
float variable = 4.44f;
```

## for

Ce mot introduit une boucle for (tant que condition, on itère), c'est une boucle dont la particularité est la possibilité de mettre des instructions dans la déclaration de la boucle. Cette déclaration se forme de 3 parties, séparées par des ;. La première partie permet de déclarer la variable d'itération, la deuxième est la condition et la dernière permet d'incrémenter notre variable d'itération.

```
for(int i = 0; i < 10; i++){  
    //Opérations  
}
```

Il existe aussi une autre forme pour cette boucle, apparue depuis la version 5.0 de Java. Cette boucle for étendue permet de parcourir une collection facilement :

```
MonObjet[] objets = new MonObjet[x];  
//...  
for(MonObjet objet : objets){  
    objet.faisQuelqueChose();  
}
```

**En savoir plus sur la boucle for étendue. [Article sur les nouveautés de Java 5.0 : La nouvelle boucle for](#)**

## goto

Ce mot-clé n'est pas utilisé actuellement.

## if

Cet opérateur permet de définir un bloc de code qui ne sera exécuté que si une certaine condition est remplie.

```
if(i == 1){ //Condition  
    i = 3; //Code exécuté seulement si i est égal à 1
```

```
}
```

## implements

Ce mot-clé est utilisé dans la définition d'une classe pour définir que la classe implémente les fonctionnalités d'une interface. Dès le moment où une classe implémente une interface, elle doit proposer une implémentation de toutes les méthodes de cette interface.

```
public class JavaTester implements Tester{  
    public boolean test(){//Méthodes définie dans l'interface  
        //Quelque chose  
    }  
}
```

## import

Ce mot clé permet de faire un raccourci sur les noms de classes en définissant le nom de package. Ainsi, on n'a pas besoin d'écrire l'entier du chemin vers la classe, mais seulement le nom de la classe.

On peut donc faire :

```
import com.test.Tester;  
//...  
Tester.test();
```

Depuis Java 5.0, on a aussi droit à l'import static, qui permet d'importer directement dans notre code, une classe ou une variable d'une classe, ceci à des fins d'allègement de code :

```
import static java.lang.Math.*;  
  
public Class Test {  
    public void calcul (int i) {  
        round(cos(i*(PI/6)-PI/2)*E);  
    }  
}
```

### En savoir plus sur l'import static. [Articles sur les nouveautés de Java 5.0 : L'import static](#)

Alors que si on n'avait pas importé la classe, on aurait du faire :

```
com.test.Tester.test();
```

## instanceof

Cette instruction de test permet de vérifier de quelle classe est un objet. Si l'objet est bien de la classe demandée, l'instruction va nous retourner true, sinon false.

```
if(monObjet instanceof Reader){  
    System.out.println("monObjet est de la classe Reader");  
}
```

## int

int est un type primitif de Java. Il peut avoir comme valeur un entier signé représenté par 32 chiffres binaires. Ce mot peut désigner un type de variable, de retour d'une méthode ou un paramètre.

```
int variable = 3;
```

## interface

Ce mot-clé permet de définir une interface. Une interface permet de définir une fonctionnalité ou un comportement qui est le même pour plusieurs classes. Une interface ne contient aucun code concret, juste des définitions de méthodes. Toutes les classes qui vont ensuite implémenter cette interface devront en redéfinir les méthodes.

```
public interface Tester(){  
    public void test();  
}
```

### En savoir plus.

## long

long est un type primitif de Java. Il peut avoir comme valeur un entier signé représenté par 64 chiffres binaires. Ce mot peut désigner un type de variable, de retour d'une méthode ou un paramètre.

```
long variable = 33;
```

## native

Ce mot-clé est utilisé dans la déclaration d'une méthode pour indiquer que cette méthode n'est pas codée en Java, mais dans un langage natif dans un fichier à part. Une telle méthode n'a donc pas de corps puisque le code ne se trouve pas dans la classe Java.

```
public native void ouvrirLecteurCD();
```

## new

Cet opérateur permet d'instancier un nouvel objet d'une classe. Il va appeler le constructeur de la classe et va construire un nouvel objet.

```
Object objet = new Object();
```

## null

null est une valeur spéciale, indiquant que la référence objet ne pointe pas sur un objet mais sur "rien". On dit d'une telle référence qu'elle est nulle.

```
String chaine = null;
```

## package

Ce mot-clé est utilisé tout en début de classe pour indiquer de quel package fait partie cette classe.

```
package com.monPackage;  
  
public class MaClasse {  
  
}
```

## private

Ce mot-clé s'utilise dans la déclaration de variables, méthodes ou classes. Quand on les déclare private, ces attributs ne sont accessibles que par la classe dans laquelle ils sont définis. Il est conseillé de déclarer privés tous les attributs de la classe.

```
private int variable = 11;
```

**Plus d'infos.**

## protected

Ce mot-clé s'utilise dans la déclaration de variables, méthodes ou classes. Quand on les déclare protected, ces attributs ne sont accessibles que par les classes filles et classes du même package.

```
protected int variable = 11;
```

**Plus d'infos.**

## public

Ce mot-clé s'utilise dans la déclaration de variables, méthodes ou classes. Quand on les déclare public, ces attributs sont accessibles par tout objet. Il faut garder à l'esprit que la classe principale d'un fichier .java doit toujours être publique.

```
public int variable = 11;
```

**Plus d'infos.**

## return

Ce mot-clé permet de sortir d'une méthode. On peut l'utiliser sans rien d'autres pour sortir d'une méthode qui ne retourne rien ou alors, on peut l'utiliser suivi du nom de l'objet qui sera retourné par la méthode. Une méthode qui retourne quelque chose doit obligatoirement posséder un return, mais une méthode sans retour n'a pas besoin de return, il peut néanmoins être utilisé pour sortir de la méthode avant la fin.

```
public int getInt(){  
    return 1;  
}  
  
public void getNothing(){  
    return;  
}
```

## short

short est un type primitif de Java. Il peut avoir comme valeur un entier signé représenté par 16 chiffres binaires. Ce mot peut désigner un type de variable, de retour d'une méthode ou un paramètre.

```
short variable = 1;
```

## static

Ce mot clé peut être utilisé pour dans la déclaration d'une variable, d'une méthode ou devant un bloc de code.

Devant une méthode ou une variable, static indique que ce membre n'appartient pas à une instance particulière de la classe. On peut donc les utiliser sans instance.

```
//Accès à la méthode statique abs de la class Math :  
Math.abs(-1);
```

Bien entendu, une méthode statique étant indépendante de l'instance, n'a pas accès aux variables non statiques de la classe.

Devant un bloc de code, on indique ce bloc sera exécuté lors du chargement de la classe.

```
public static ArrayList<String> liste = new ArrayList<String>();  
static {  
    liste.add(new String("x"));  
    liste.add(new String("y"));  
}
```

A noter que depuis Java 5.0, on peut aussi utiliser le mot-clé static pour un import. Voir le mot-clé **import** pour plus d'infos.

### Plus d'infos

## strictfp

Ce mot-clé est utilisable sur la déclaration de classes, d'interfaces ou de méthodes. Il oblige la JVM à effectuer les calculs selon les spécifications du langage, donc il permet de garantir les mêmes calculs quelle que soit la machine virtuelle sur laquelle l'opération est effectuée.

### Plus d'infos.

Ce mot-clé a été rajouté dans la version 1.2 de Java.

## super

Ce mot-clé est une référence sur la classe mère, c'est à dire la classe dont hérite la classe en cours.

```
public MaClasse() {
```

```
super(); //On appelle le constructeur de la classe mère
}
public void maMethode(){
    super.maMethode();//On appelle la méthode de classe mère
}
```

## Plus d'infos.

### switch

Ce mot-clé introduit une instruction de contrôle. Cette instruction teste la valeur d'une variable et en fonction de sa valeur effectue le traitement correspondant. On définit donc des étiquettes avec des valeurs et des actions qui vont s'effectuer si la valeur testée est égale à la valeur de l'étiquette.

```
switch(valeur){
    case 1 :
        //Actions si valeur vaut 1
        break;
    case 2 :
        //Actions si valeur vaut 2
        break;
    case default :
        //Actions dans les autres cas
}
```

### synchronized

Ce mot-clé est utilisé dans la programmation multithread dans la déclaration d'une méthode ou d'un bloc d'instructions pour indiquer que seul un thread peut accéder en même temps à ce bloc ou à cette méthode

```
public void synchronized methode(){
    System.out.println("Deux thread ne peuvent pas appeler cette méthode en même temps");
}
```

ce qui est équivalent d'ailleurs à :

```
public void methode(){
    synchronized(this){
        System.out.println("Deux thread ne peuvent pas appeler cette méthode en même temps");
    }
}
```

Si la méthode est static, voici son équivalent :

```
public class MaClass {
    public static void methode() {
        synchronized(MaClass.class) {
            // code
        }
    }
}
```

Utilisé sur un objet, voici ce que ça donne :

```
synchronized(monObjet) {
    // code
}
```

## this

Ce mot-clé est une référence sur l'objet en cours, c'est à dire la classe dans laquelle on se trouve.

```
public MaClasse(String uneValeur){
    super();
    this.attribut = attribut; //On prend attribut de la classe
    this.refreshAttributes(); //On appelle une méthode de la classe
}
/** On peut aussi accéder aux constructeurs de la classe elle-même*/
public MaClasse(String attribut){
    this("une valeur par défaut");
    //on appelle ici le constructeur défini un peu plus haut
}
```

### Plus d'infos.

## throw

Cette instruction permet de lever une nouvelle exception. La classe de l'exception doit obligatoirement descendre de **Throwable**. Dès qu'une exception est levée, la main est directement repassée à la méthode appelante, donc le code après le throw n'est jamais exécuté.

```
if(erreur) throw new Exception("Une erreur est arrivée");
```

## throws

Ce mot-clé est utilisé dans la déclaration d'une méthode pour indiquer que celle-ci peut lever des exceptions qui ne sont pas traitées.

```
public void read throws IOException {
    //Code pouvant lancer une IOException
}
```

## transient

Ce mot clé permet de ne pas sauvegarder une variable lors de la sérialisation de la classe. Ainsi la variable ne va pas apparaître dans le fichier.

```
public transient int variable = 2;
```

### Plus d'infos. Article sur la sérialisation : Le mot-clé transient

## true

Ce mot représente une valeur du type boolean, elle indique une condition remplie (vraie).

```
boolean variable = true;
```

## try

Ce mot-clé introduit un bloc d'instructions. Il n'a pas d'autres utilisés que celle de permettre l'utilisation de bloc catch et/ou finally.

```
try{
  //Instructions
} finally {
  //Instructions
}
```

## void

Ce mot clé s'utilise dans la déclaration du type de retour d'une méthode. Il indique que la méthode ne retourne rien.

```
public void methodeQuiRetourneRien(){
  //Instructions diverses
}
```

## volatile

On utilise ce mot-clé sur des variables modifiables de manière asynchrone, donc plus d'un thread peuvent y accéder en même temps.

Le fait d'employer ce mot-clé oblige la JVM à rafraîchir son contenu à chaque utilisation. Ainsi, on est sûr qu'on n'accède pas à la valeur mise en cache mais bien à la valeur correcte.

```
public volatile Integer = 5;
```

## Plus d'infos.

## while

Ce mot introduit une boucle tant que. C'est à dire que les instructions contenues dans le while sont exécutées tant que la condition du while est remplie. Cette condition est vérifiée avant de rentrer dans la boucle, il est donc possible de ne pas entrer dans la boucle si la condition n'est pas remplie.

```
while(condition){
  //Instructions
}
```

## IV - Remerciements

Je tiens à remercier **adiGuba**, **vbrabant** et **Ricky81** pour les corrections et leurs remarques.

