

# Qualité de développement

CM1 : Rappels, Java, bases

Mickaël Martin Nevot

V2.0.0



Cette œuvre de [Mickaël Martin Nevot](#) est mise à disposition selon les termes de la [licence Creative Commons Attribution – Pas d'Utilisation Commerciale – Partage à l'Identique 3.0 non transposé](#).

# Qualité de développement

- I. Prés.
- II. Java bas.
- III. Obj.
- IV. Hérit.
- V. POO
- VI. Excep.
- VII. Poly.
- VIII. Thread
- IX. Java av.
- X. Algo. av.
- XI. APP
- XII. GL

# Java

- Sun Microsystem (1995)
- Langage :
  - **Orienté objet et fortement typé**
  - **Héritage simple**, interface, polymorphisme
- JRE :
  - JVM : **machine virtuelle** qui interprète le code
  - API : bibliothèques standards
- JDK :
  - Compilateur
  - JVM : débogueur



# Philosophie

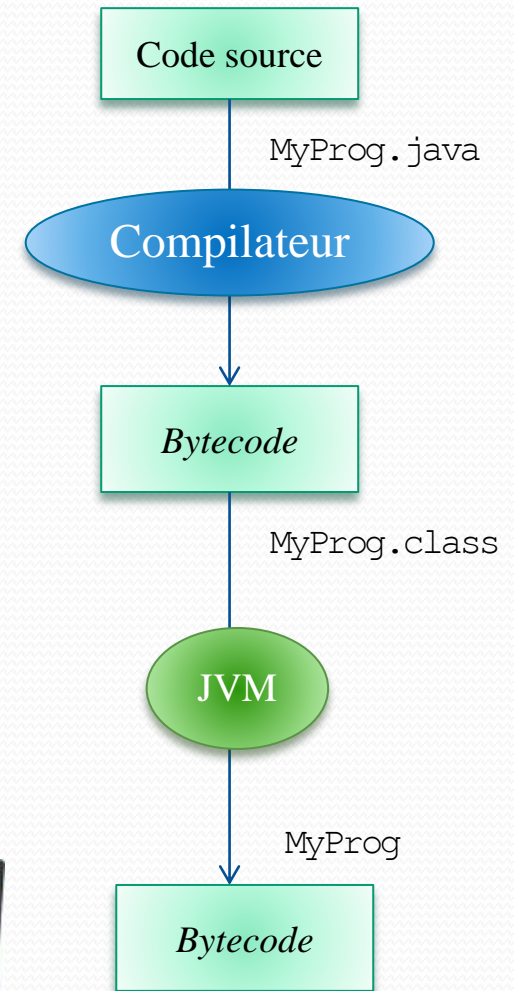
- **Simple** et familier
- **Robuste** et sûr
- **Indépendant** de la machine employée pour l'exécution
- Très **performant**
- Interprété, multitâches et dynamique
- Pourquoi apprendre Java ?
  - Plus de 4,5 milliards de périphériques
  - Programmes Web et services Web
  - Programme sur téléphone portable

Duke, la mascotte de Java

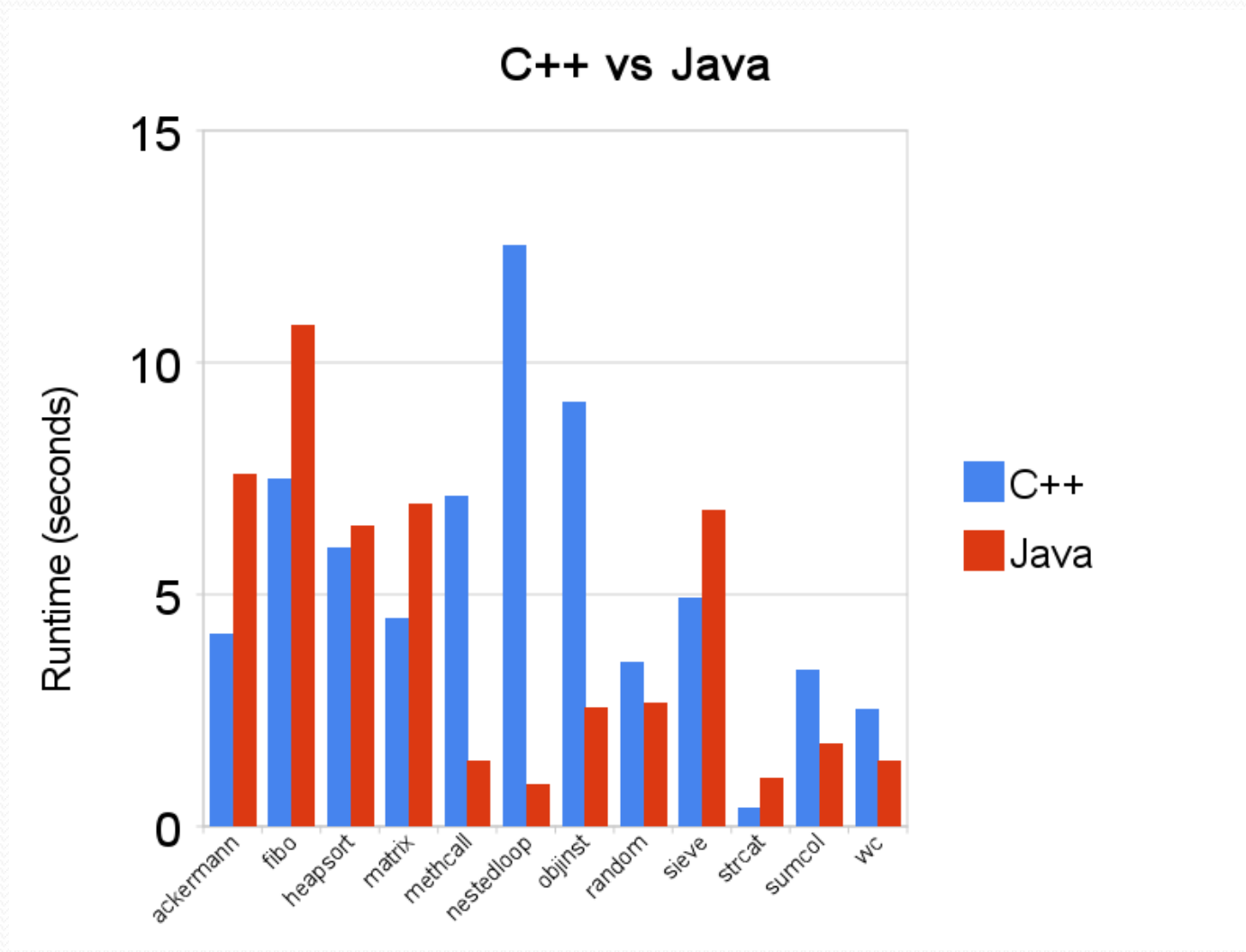


# Fonctionnement

- Création du code source
- **Compilation** en *bytecode* :
  - À partir du code source
  - Code exécutable sur toute JVM
- Exécution :
  - Interprète le *bytecode*
  - *Bytecode* indépendant de la plateforme



# Différences par rapport à C++



# Structure du code source

- Un fichier source Java contient une **classe**
- Une classe contient des **attributs** et des **méthodes**
- Une méthode contient des **instructions**

```
// Déclaration de classe.  
class MyClass {  
    // Déclaration d'attribut.  
    int att1;  
  
    //Déclaration de méthode.  
    void meth1(int i) {  
        instruction1  
        instruction2  
        ...  
    }  
}
```

# Structure du code source

- Méthode :

typeDeRetour myMeth(paramètre(s))

Signature

L'ordre des paramètres est déterminant

Délimitation de la classe

// Déclaration de classe.

```
class MyClass {
```

```
    int att1; // Déclaration d'attribut.
```

```
    float meth1(int i) { //Déclaration de méthode.
```

```
        instruction1
```

```
        instruction2
```

```
        ...
```

```
    }
```

```
}
```

Paramètre (typé)

Délimitation de la méthode

Valeur de retour  
(void : aucune)



# Utilisation

- Fichier source : extension `.java`
- Fichier binaire : extension `.class`
- **Un** fichier source contient **une** classe
- Le nom du fichier est identique au nom de la classe
- On lance l'exécution par la classe principale :
  - Contient un point de commencement d'exécution du code
- Sensible à la casse : `MyClass` est différent de `Myclass`
- Respecter les mots-clefs réservés

# Instruction

- Se termine par ;
- Type d'instruction :
  - Déclaration : `String att1;`
  - Affectation : `a = 10;`
  - Appel de fonction/méthode : `myMeth( );`
  - Instruction conditionnelle : `if (a == b) { ... }`
  - Instruction vide : ;
  - Bloc (d'instructions) :

```
{  
    instruction1  
    ...  
}
```

# Mise en forme / commentaires

- Mise en forme :
  - Indentation à chaque niveau de bloc
  - Convention de nommage :
    - mypackage
    - MyClass
    - myMethod
    - myVar
    - MY\_CONST

- Commentaires de type C/C++ :

```
// Commentaire (une seule ligne)
/* Autre commentaire (une ligne). */
/*
   Autre commentaire (sur plusieurs lignes).
*/
```

# Structure de contrôle

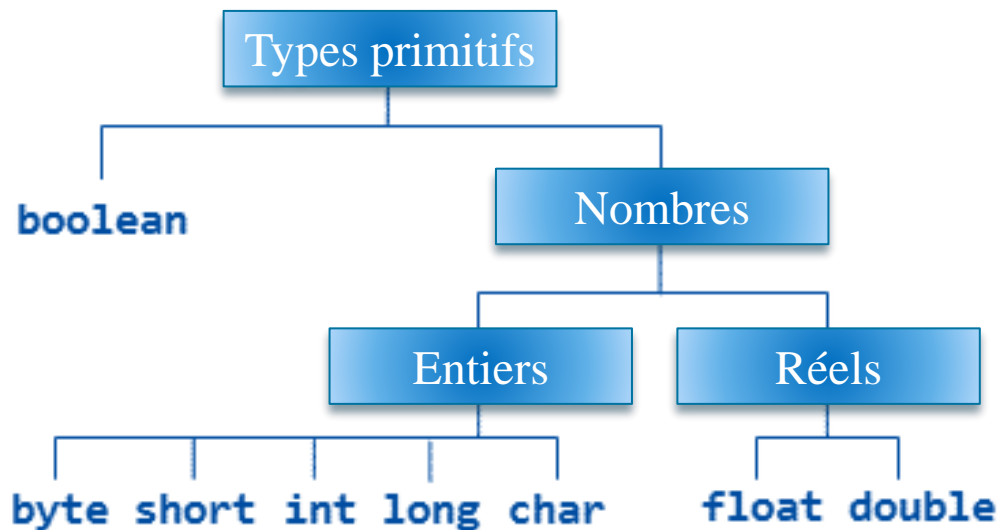
- Conditionnelle :
  - `if (condition) { ... } else { ... }`
- Branchement conditionnel :
  - `switch (ident) { case val0 : ... case val1 : ... default: ... }`
- Boucles :
  - `for (initialisation ; condition ; modification) { ... }`
  - `for (Type var : Collection) { ... }`
  - `while (condition) { ... }`
  - `do { ... } while (condition)`
- Mots clefs break/continue :
  - `break` : permet de sortir du bloc (boucle, branchement conditionnel, etc.)
  - `continue` : « saute » à l'itération suivante d'une boucle

# Portée et variable locale

- **Portée** (d'une variable) :
  - Début : à partir de sa déclaration
  - Fin : la fin du bloc d'instructions dans lequel elle se trouve (ou celui du corps de la méthode pour un paramètre)
- **Variable locale** :
  - Déclarée dans une méthode ou un bloc d'une méthode
  - Durée de vie : sa portée
  - **Visible qu'à l'intérieur du bloc**
  - Pas de valeur par défaut

# Type primitif

- N'est pas un objet
- Occupe une place fixe en mémoire
- Dispose d'un *alter ego* objet et d'une méthode de conversion
- Est converti automatiquement en référence (*autoboxing*)
- Conversion de type explicite (*cast*) : (`type`)



# Types primitifs

- Entiers :

• <code>byte</code>	-128 à 127	1 octet
• <code>short</code>	-32768 à 32768	2 octets
• <code>int</code>	-2147483648 à 2147483647	4 octets
• <code>long</code>	-9223372036854775808 à 9223372036854775807	8 octets

- Flottants :

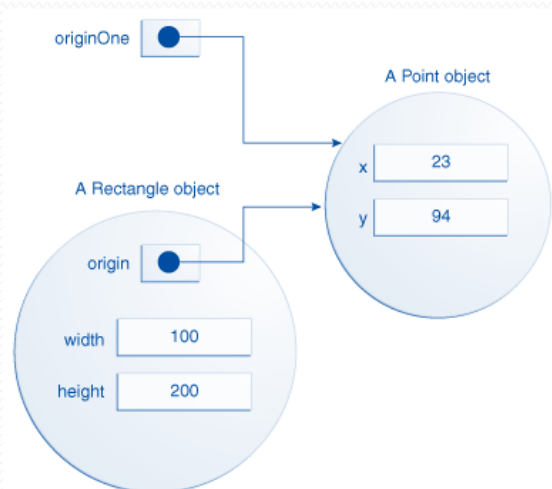
• <code>float</code>	variable	4 octets
• <code>double</code>	variable	8 octets

- Autres :

• <code>boolean</code>	<code>true</code> , <code>false</code>	selon la JVM
• <code>char</code> (Unicode, c.-à-d. a)	0 à 65535	2 octets

# Référence

- Référence vers un objet : il n'existe **pas de variable objet**
- Une référence déclarée pour un type d'objet ne peut référencer que des objets de ce type
- Une référence ne référence qu'un seul objet à la fois
- Un objet peut être référencé par plusieurs références
- Aucune référence : `null`





# Variable et constante

- Variable (deux catégories : **primitive** ou **référence**) :
  - Identifiant
  - Type
- Constante :
  - Variable ne pouvant avoir qu'une seule affectation
  - **Non modifiable**
  - Mot clef `final`

```
final int n = 5;  
final int t;  
...  
t = 8;  
n = 10; // Erreur !
```



# Tableau

- Considéré comme un objet
- Un seul type par tableau (primitif/objet)
- Indices commencent à zéro
- Mot clef `new` :
  - Alloue la mémoire en fonction de la taille (fixe)
  - Initialise à 0 (type primitif)
- Multidimensionnel (tableau de type tableau) :

Pas de dimensions à la déclaration

```
int[] myTab; // Déclaration.  
myTab = new int[3]; // Dimensionnement.  
myTab[0] = 1;  
myTab[2] = 5;
```

```
int myTab[] = {1, 0, 5};
```

=

# Opérateurs

- Unaires :
  - Arithmétiques : +, -
  - Incrémentation/Décrémentation (pré, post) : ++, --
  - Transtypage : (`type`)
- Binaires :
  - Arithmétiques : +, -, \*, /, %
  - Affectations (élargies) : =, +=, -=, \*=, /=
  - Comparaisons : ==, >, >=, <, !=
  - Logiques : &&, ||, &, |
  - Concaténation : +

# Classe Object / méthode main

- `Object` :
  - **Classe de plus haut niveau** dans la hiérarchie d'héritage
  - Toute classe autre que `Object` possède une super-classe
  - Toute classe hérite directement ou pas de `Object`
  - Toute classe qui n'a pas de clause `extends` hérite de `Object`
- `main( ... )` :
  - Point de commencement d'exécution du code
  - Au moins une par application (**classe principale**) :

```
public static void main(String[] args) {  
    // Le code va commencer par s'exécuter ici.  
}
```

# A savoir

- Classe String :

```
String myString = "Hello!";  
myString += "How are you?";
```

- Classe File :

```
File myFile = new File("file.txt");
```

- Affichage (y compris types primitifs/références) :

```
System.out.println("a = " + a);
```



# Javadoc

- Outil standard pour créer une **documentation** d'API
- Génération automatique en HTML
- Utilisation ( $\neq$  commentaire `/* */`) :
  - Première ligne : uniquement `/**`
  - Lignes suivantes : un espace suivi de `*`
  - Dernière ligne : un espace suivi uniquement de `*/`
  - L'entité documentée est précédée par son commentaire
  - Tags prédéfinis

Bonne utilisation : expliquer n'est pas traduire !

# Javadoc : principaux tags

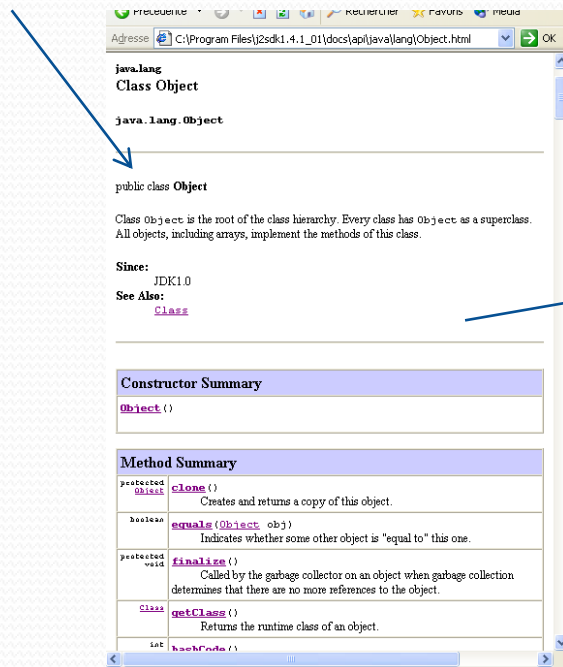
- `@author` : nom du développeur
- `@version` : version d'une classe/méthode
- `@param` : définit un paramètre de méthode :  
requis pour chaque paramètre
- `@since` : version du JDK de l'apparition de la classe/méthode
- `@return` : valeur de retour
- `@throws` : classe de l'exception et conditions de lancement
- `@deprecated` : marque la méthode comme dépréciée
- `@see` : référence croisée avec un autre élément

# Exemple de Javadoc

```
/**
```

- \* Valide un mouvement de jeu d'Échecs.
- \* @param beginCol Colonne de la case de départ
- \* @param beginRow Ligne de la case de départ
- \* @param endCol Colonne de la case de destination
- \* @param endRow Ligne de la case de destination
- \* @return vrai(true) si le mouvement d'échec est valide ou faux(false) sinon

```
*/
```



java.lang.  
Class Object

java.lang.Object

---

public class Object

Class Object is the root of the class hierarchy. Every class has Object as a superclass. All objects, including arrays, implement the methods of this class.

Since:  
JDK1.0

See Also:  
[Class](#)

---

**Constructor Summary**

[Object](#) ()

---

**Method Summary**

Modifier	Method	Description
protected	<a href="#">clone</a> ()	Creates and returns a copy of this object.
boolean	<a href="#">equals</a> (Object obj)	Indicates whether some other object is "equal to" this one.
protected	<a href="#">finalize</a> ()	Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
Class	<a href="#">getClass</a> ()	Returns the runtime class of an object.
int	<a href="#">hashCode</a> ()	

**Method Detail**

**getClass**

```
public final Class getClass ()
```

Returns the runtime class of an object. That `Class` object is the object that is locked by `static synchronized` methods of the represented class.

**Returns:**  
the object of type `Class` that represents the runtime class of the object.

---

**hashCode**

```
public int hashCode ()
```

Returns a hash code value for the object. This method is supported for the benefit of hash tables such as those provided by `java.util.Hashtable`.

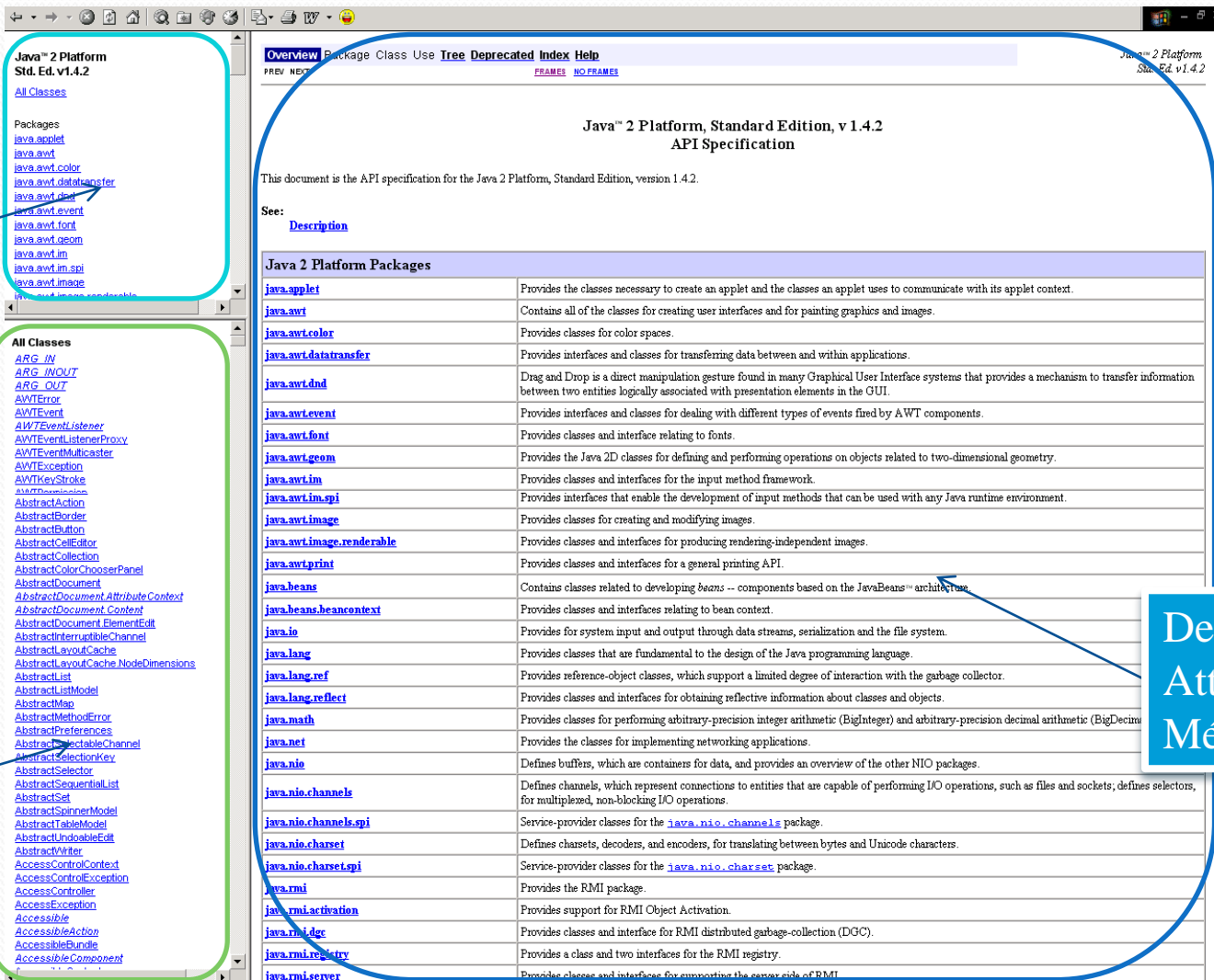
The general contract of `hashCode` is:

- Whenever it is invoked on the same object more than once during an execution, the returned value must be identical to the value returned by the first invocation.



# API

Paquetages



Java™ 2 Platform  
Std. Ed. v1.4.2

All Classes

Packages

- [java.applet](#)
- [java.awt](#)
- [java.awt.color](#)
- [java.awt.datatransfer](#)
- [java.awt.dnd](#)
- [java.awt.event](#)
- [java.awt.font](#)
- [java.awt.geom](#)
- [java.awt.im](#)
- [java.awt.im.spi](#)
- [java.awt.image](#)
- [java.awt.image.renderable](#)
- [java.awt.print](#)
- [java.beans](#)
- [java.io](#)
- [java.lang](#)
- [java.lang.ref](#)
- [java.lang.reflect](#)
- [java.math](#)
- [java.net](#)
- [java.nio](#)
- [java.nio.channels](#)
- [java.nio.channels.spi](#)
- [java.nio.charset](#)
- [java.nio.charset.spi](#)
- [java.rmi](#)
- [java.rmi.activation](#)
- [java.rmi.dgc](#)
- [java.rmi.registry](#)
- [java.rmi.server](#)

All Classes

- [ARG\\_IN](#)
- [ARG\\_INOUT](#)
- [ARG\\_OUT](#)
- [AWTError](#)
- [AWTEvent](#)
- [AWTEventListener](#)
- [AWTEventListenerProxy](#)
- [AWTEventMulticaster](#)
- [AWTException](#)
- [AWTKeyStroke](#)
- [AWTMouseEvent](#)
- [AbstractAction](#)
- [AbstractBorder](#)
- [AbstractButton](#)
- [AbstractCellEditor](#)
- [AbstractCollection](#)
- [AbstractColorChooserPanel](#)
- [AbstractDocument](#)
- [AbstractDocument.AttributeContext](#)
- [AbstractDocument.Content](#)
- [AbstractDocument.ElementEdit](#)
- [AbstractInterruptibleChannel](#)
- [AbstractLayoutCache](#)
- [AbstractLayoutCache.NodeDimensions](#)
- [AbstractList](#)
- [AbstractListModel](#)
- [AbstractMap](#)
- [AbstractMethodError](#)
- [AbstractPreferences](#)
- [AbstractSelectableChannel](#)
- [AbstractSelectableKey](#)
- [AbstractSelector](#)
- [AbstractSequentialList](#)
- [AbstractSet](#)
- [AbstractSpinnerModel](#)
- [AbstractTableModel](#)
- [AbstractUndoableEdit](#)
- [AbstractWriter](#)
- [AccessControlContext](#)
- [AccessControlException](#)
- [AccessController](#)
- [AccessController](#)
- [AccessException](#)
- [Accessible](#)
- [AccessibleAction](#)
- [AccessibleBundle](#)
- [AccessibleComponent](#)

Overview Package Class Use Tree Deprecated Index Help

PREV NEXT

FRAMES NO FRAMES

## Java™ 2 Platform, Standard Edition, v 1.4.2 API Specification

This document is the API specification for the Java 2 Platform, Standard Edition, version 1.4.2.

See:

- [Description](#)

### Java 2 Platform Packages

<a href="#">java.applet</a>	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
<a href="#">java.awt</a>	Contains all of the classes for creating user interfaces and for painting graphics and images.
<a href="#">java.awt.color</a>	Provides classes for color spaces.
<a href="#">java.awt.datatransfer</a>	Provides interfaces and classes for transferring data between and within applications.
<a href="#">java.awt.dnd</a>	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
<a href="#">java.awt.event</a>	Provides interfaces and classes for dealing with different types of events fired by AWT components.
<a href="#">java.awt.font</a>	Provides classes and interface relating to fonts.
<a href="#">java.awt.geom</a>	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.
<a href="#">java.awt.im</a>	Provides classes and interfaces for the input method framework.
<a href="#">java.awt.im.spi</a>	Provides interfaces that enable the development of input methods that can be used with any Java runtime environment.
<a href="#">java.awt.image</a>	Provides classes for creating and modifying images.
<a href="#">java.awt.image.renderable</a>	Provides classes and interfaces for producing rendering-independent images.
<a href="#">java.awt.print</a>	Provides classes and interfaces for a general printing API.
<a href="#">java.beans</a>	Contains classes related to developing <i>beans</i> -- components based on the JavaBeans™ architecture.
<a href="#">java.beans.beancontext</a>	Provides classes and interfaces relating to bean context.
<a href="#">java.io</a>	Provides for system input and output through data streams, serialization and the file system.
<a href="#">java.lang</a>	Provides classes that are fundamental to the design of the Java programming language.
<a href="#">java.lang.ref</a>	Provides reference-object classes, which support a limited degree of interaction with the garbage collector.
<a href="#">java.lang.reflect</a>	Provides classes and interfaces for obtaining reflective information about classes and objects.
<a href="#">java.math</a>	Provides classes for performing arbitrary-precision integer arithmetic (BigInteger) and arbitrary-precision decimal arithmetic (BigDecimal).
<a href="#">java.net</a>	Provides the classes for implementing networking applications.
<a href="#">java.nio</a>	Defines buffers, which are containers for data, and provides an overview of the other NIO packages.
<a href="#">java.nio.channels</a>	Defines channels, which represent connections to entities that are capable of performing I/O operations, such as files and sockets; defines selectors, for multiplexed, non-blocking I/O operations.
<a href="#">java.nio.channels.spi</a>	Service-provider classes for the <a href="#">java.nio.channels</a> package.
<a href="#">java.nio.charset</a>	Defines charsets, decoders, and encoders, for translating between bytes and Unicode characters.
<a href="#">java.nio.charset.spi</a>	Service-provider classes for the <a href="#">java.nio.charset</a> package.
<a href="#">java.rmi</a>	Provides the RMI package.
<a href="#">java.rmi.activation</a>	Provides support for RMI Object Activation.
<a href="#">java.rmi.dgc</a>	Provides classes and interface for RMI distributed garbage-collection (DGC).
<a href="#">java.rmi.registry</a>	Provides a class and two interfaces for the RMI registry.
<a href="#">java.rmi.server</a>	Provides classes and interfaces for supporting the server side of RMI.

Description  
Attributs  
Méthodes

Classes

# Outils

- Éditeur :
  - Eclipse : <http://www.eclipse.org>
- Ressources Java :
  - API :  
<http://download.oracle.com/javase/1.5.0/docs/api>
  - Convention de nommage :  
<http://java.sun.com/docs/codeconv/CodeConventions.pdf>
  - Mots clefs réservés :  
[http://download.oracle.com/javase/tutorial/java/nu  
tsandbolts/\\_keywords.html](http://download.oracle.com/javase/tutorial/java/nu<br/>tsandbolts/_keywords.html)



# Bonnes pratiques

- Penser à l'initialisation pour éviter une erreur
- Penser à construire les objets avant de les utiliser
- Penser à l'utilisation de `break` dans un `switch`
- Attention à l'encapsulation
- Utiliser le mot clef `this` autant de fois que possible
- Pas de mot clef `then` (en relation avec un `if`)
- Pas d'utilisation de variable d'instance ni du mot clef `this` dans une méthode de classe
- Pas d'héritage multiple

# Crédits

## Auteur

Mickaël Martin Nevot

[mmartin.nevot@gmail.com](mailto:mmartin.nevot@gmail.com)



Carte de visite électronique

## Relecteurs

Cours en ligne sur : [www.mickaël-martin-nevot.com](http://www.mickaël-martin-nevot.com)

