Qualité de développement

CM1 : Rappels, Java, bases

Mickaël Martin Nevot

V2.0.1

Cette œuvre de Mickaël Martin Nevot est mise à disposition sous licence Creative Commons Attribution - Utilisation non commerciale - Partage dans les mêmes conditions.

BY-NC-SA

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

 BY-NC-SA

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

BY-NC-SA

Philosophie

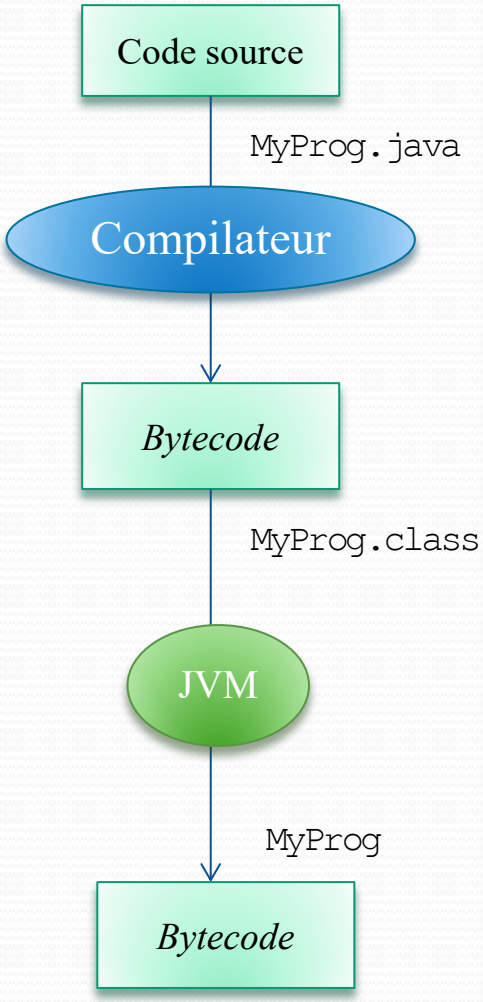
- **Simple** et familier
- **Robuste** et sûr
- **Indépendant** de la machine employée pour l'exécution
- Très **performant**
- Interprété, multitâches et dynamique
- Pourquoi apprendre Java ?
  - Plus de 4,5 milliards de périphériques
  - Programmes Web et services Web
  - Programme sur téléphone portable

Duke, la mascotte de Java

(cc) BY-NC-SA

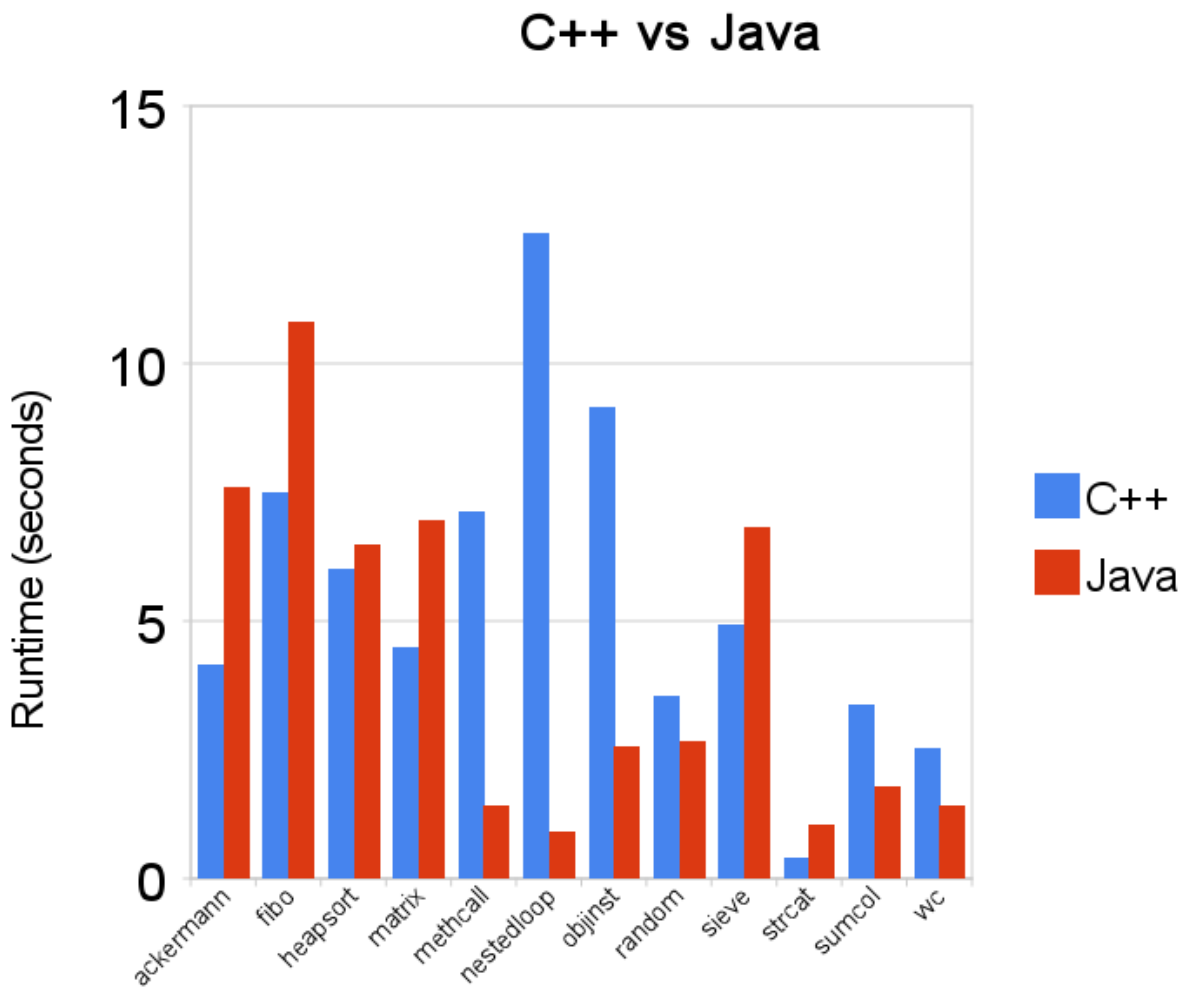
Fonctionnement

- 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



(cc) BY-NC-SA

Différences par rapport à C++



(cc) BY-NC-SA

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
        …
    }
}
```

(cc) BY-NC-SA

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)

(cc) BY-NC-SA

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

(cc) BY-NC-SA

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
    …
}
```

(cc) BY-NC-SA

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).
*/
```

 BY-NC-SA

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

 (cc) BY-NC-SA

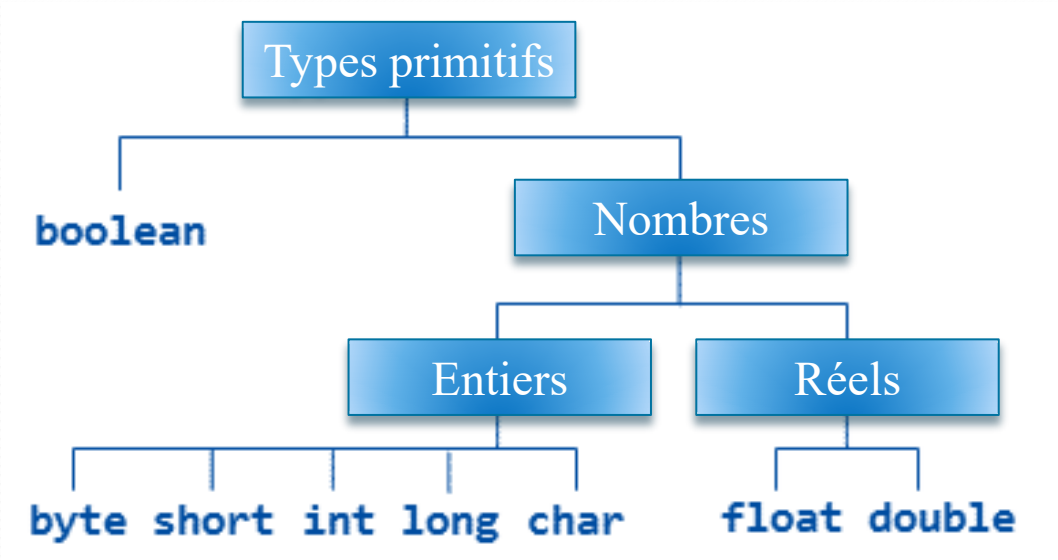
Portée et variable locale

- **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

(cc) BY-NC-SA

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)`



 (cc) BY-NC-SA

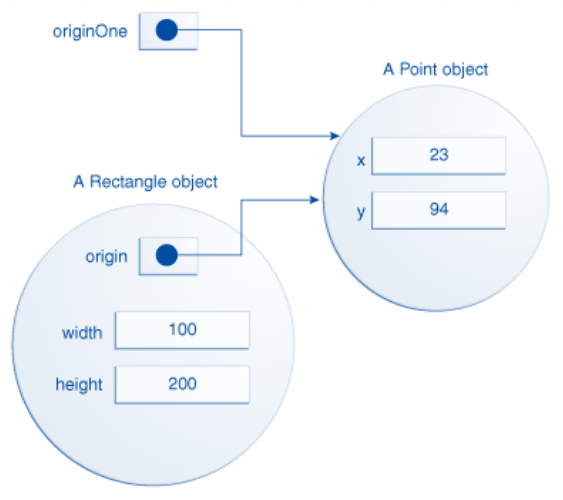
Types primitifs

- Entiers :
  - `byte` `-128` à `127` 1 octet
  - `short` `-32768` à `32768` 2 octets
  - `int` `-2147483648` à `2147483647` 4 octets
  - `long` `-9223372036854775808` à `9223372036854775807` 8 octets
- Flottants :
  - `float` variable 4 octets
  - `double` variable 8 octets
- Autres :
  - `boolean` `true`, `false` selon la JVM
  - `char` (Unicode, c.-à-d. `a`) `0` à `65535` 2 octets

(cc) BY-NC-SA

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`



 BY-NC-SA

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 !
```

(cc) BY-NC-SA

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};
```

 (cc) BY-NC-SA

Opérateurs

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

 (cc) BY-NC-SA

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.
}
```

 (cc) BY-NC-SA

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);
```

 BY-NC-SA

Javadoc

- Outil standard pour créer une **documentation** d'API
- Génération automatique en HTML
- Utilisation (≠ 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 !

 BY-NC-SA

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

BY-NC-SA

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
 */
```

Adresse C:\Program Files\j2sdk1.4.1_01\docs\api\java\lang\Object.html OK

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

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 hashtables 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

BY-NC-SA

API

Paquetages

Classes

Description
Attributs
Méthodes

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

All Classes
ARG_IN
ARG_INOUT
ARG_OUT
AWTError
AWTEvent
AWTEventListener
AWTEventListenerProxy
AWTEventMulticaster
AWTException
AWTKeyStroke
AbstractAction
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractCollection
AbstractColorChooserPanel
AbstractDocument
AbstractDocument.AttributeContext
AbstractDocument.Content
AbstractDocument.ElementEdit
AbstractInterruptibleChannel
AbstractLayoutCache
AbstractLayoutCache.NodeDimensions
AbstractList
AbstractListModel
AbstractMap
AbstractMethodError
AbstractPreferences
AbstractSelectableChannel
AbstractSelectionKey
AbstractSelector
AbstractSequentialList
AbstractSet
AbstractSpinnerModel
AbstractTableModel
AbstractUndoableEdit
AbstractWriter
AccessControlContext
AccessControlException
AccessController
AccessException
Accessible
AccessibleAction
AccessibleBundle
AccessibleComponent

Overview Package Class Use Tree Deprecated Index Help
PREV NEXT FRAMES NO FRAMES

Java™ 2 Platform
Std. Ed. v1.4.2

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	
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	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.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font	Provides classes and interface relating to fonts.
java.awt.geom	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.
java.awt.im	Provides classes and interfaces for the input method framework.
java.awt.im.spi	Provides interfaces that enable the development of input methods that can be used with any Java runtime environment.
java.awt.image	Provides classes for creating and modifying images.
java.awt.image.renderable	Provides classes and interfaces for producing rendering-independent images.
java.awt.print	Provides classes and interfaces for a general printing API.
java.beans	Contains classes related to developing *beans* -- components based on the JavaBeans™ architecture.
java.beans.beancontext	Provides classes and interfaces relating to bean context.
java.io	Provides for system input and output through data streams, serialization and the file system.
java.lang	Provides classes that are fundamental to the design of the Java programming language.
java.lang.ref	Provides reference-object classes, which support a limited degree of interaction with the garbage collector.
java.lang.reflect	Provides classes and interfaces for obtaining reflective information about classes and objects.
java.math	Provides classes for performing arbitrary-precision integer arithmetic (BigInteger) and arbitrary-precision decimal arithmetic (BigDecim
java.net	Provides the classes for implementing networking applications.
java.nio	Defines buffers, which are containers for data, and provides an overview of the other NIO packages.
java.nio.channels	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.
java.nio.channels.spi	Service-provider classes for the `java.nio.channels` package.
java.nio.charset	Defines charsets, decoders, and encoders, for translating between bytes and Unicode characters.
java.nio.charset.spi	Service-provider classes for the `java.nio.charset` package.
java.rmi	Provides the RMI package.
java.rmi.activation	Provides support for RMI Object Activation.
java.rmi.dgc	Provides classes and interface for RMI distributed garbage-collection (DGC).
java.rmi.registry	Provides a class and two interfaces for the RMI registry.
java.rmi.server	Provides classes and interfaces for supporting the server side of RMI.

 (cc) BY-NC-SA

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/nutsandbolts/_keywords.html

(cc) BY-NC-SA

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

BY-NC-SA

Crédits

Auteur

Mickaël Martin Nevot
mmartin.nevot@gmail.com

Carte de visite électronique

Relecteurs

Cours en ligne sur : www.mickael-martin-nevot.com