Qualité de développement

CM1 : Rappels, Java, bases

Mickaël Martin Nevot

V2.0.0

Qualité de développement

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

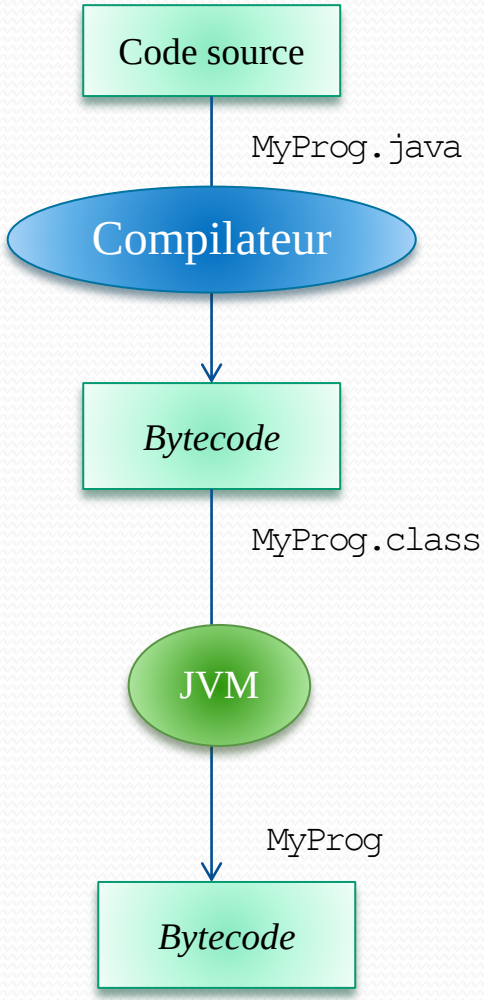
Philosophie

- **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

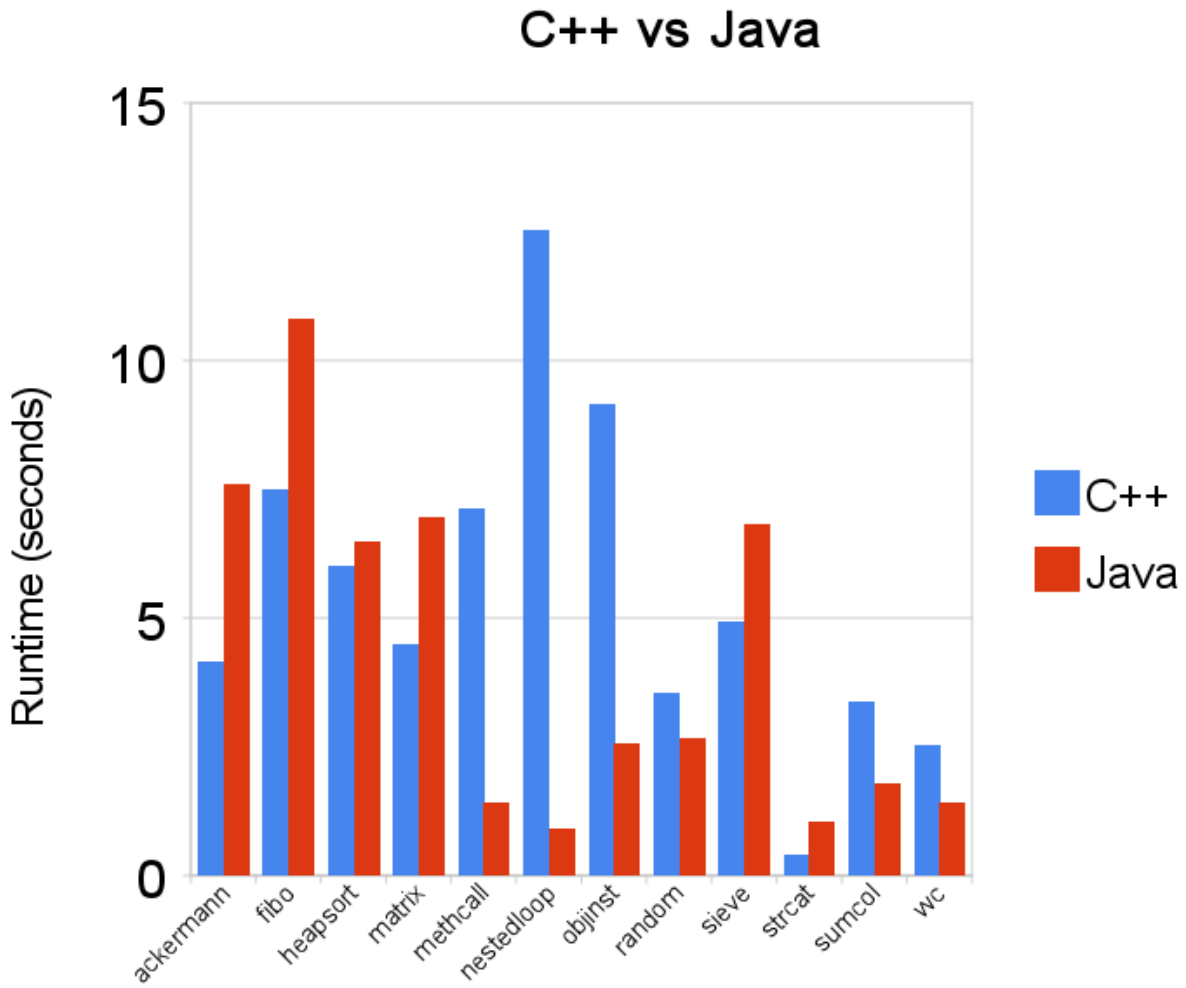
Duke, la mascotte de Java

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

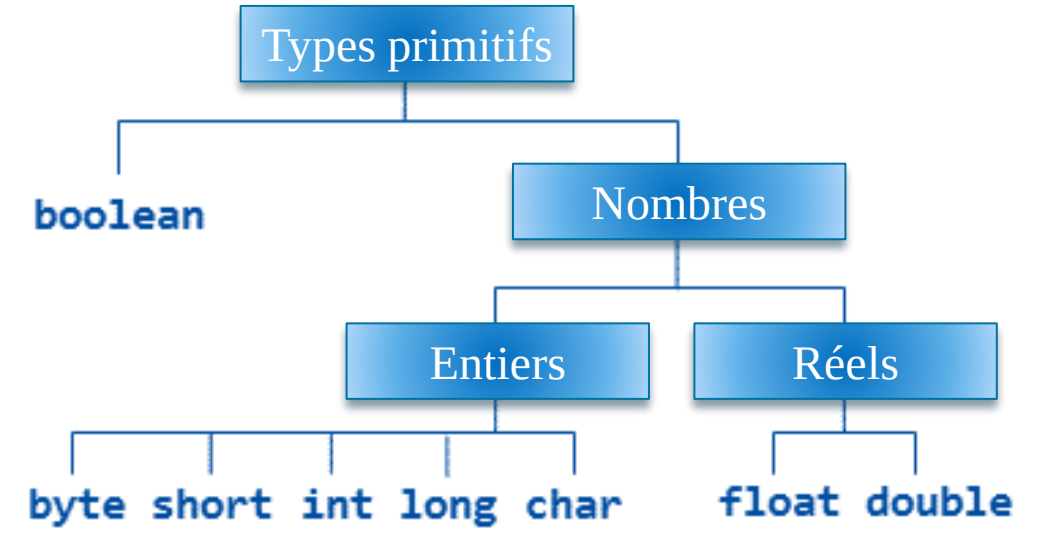
- Conditionnelle :
  - `if (condition) { … } else { … }`
- Branchement conditionnel :
  - `switch (ident) { case val0 : … case val1 : … default: … }`
- Boucles :
  - `for (initialisation ; condition ; modification) { … }`
  - `for (Type var : Collection) { … }`
  - `while (condition) { … }`
  - `do { … } 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)`

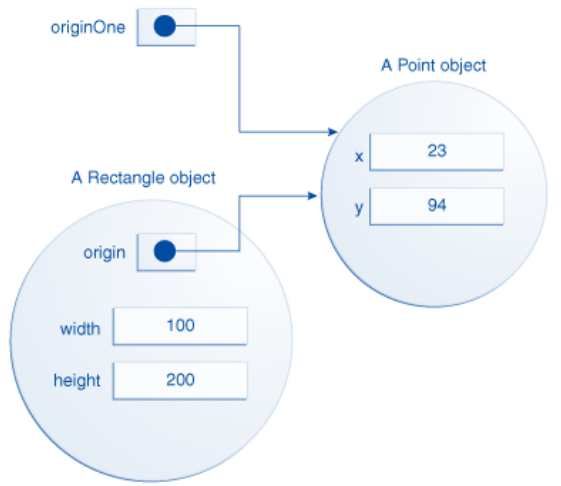


Types primitifs

- Entiers :
  - `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

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 (≠ 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

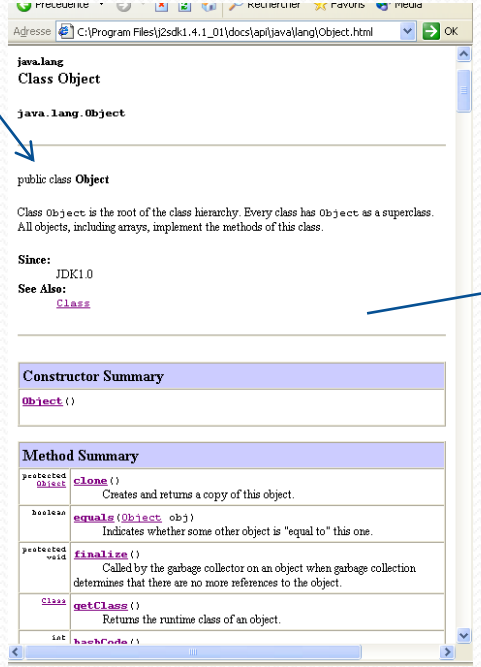
Bonne utilisation : expliquer n'est pas traduire !

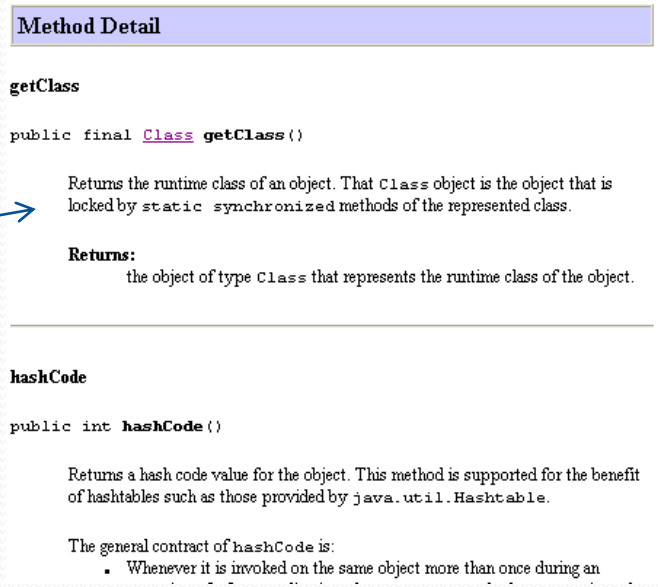
Javadoc : principaux tags

- `@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
 */
```



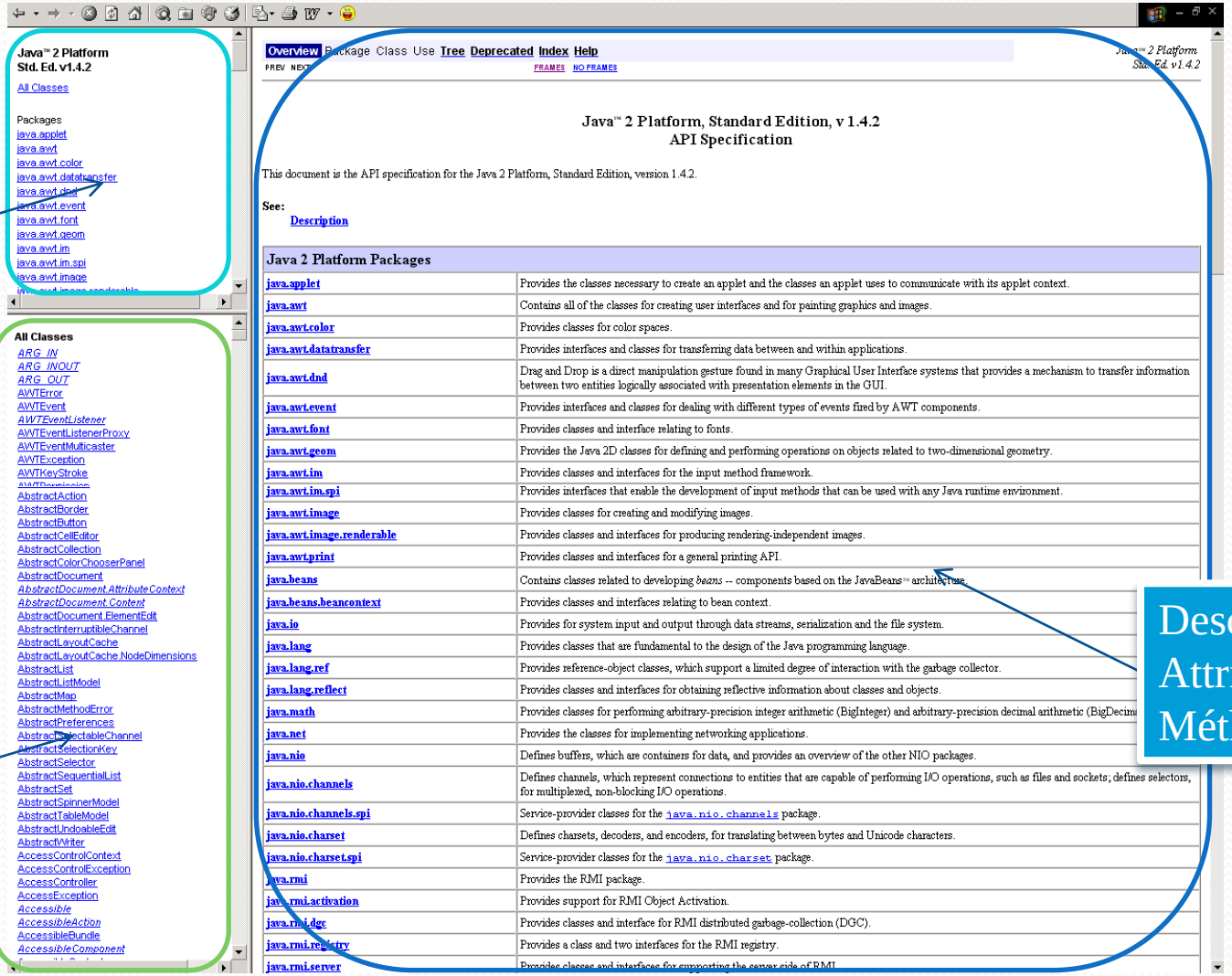


API

Paquetages

Classes

Description
Attributs
Méthodes



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

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

Carte de visite électronique

Relecteurs

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