MID0501V – Cours Java

Éric Jacoboni

13 septembre 2021

Université Toulouse – Jean Jaurès

Sommaire

Présentation du langage

Bibliographie et ressources

- *Beginning Java*, par Ivor Horton, Éd. John Wiley & Sons.
- *Java, The complete Reference*, par Herbert Schildt, Éd. Oracle Press.
- La documentation de l'API, sur le site d'Oracle.
- Blogs de développeurs.

Outils nécessaires

- Au minimum, vous avez besoin de :
  - Un éditeur de texte (de préférence avec un « mode Java »).
  - Un JDK (OpenJDK ou celui d'Oracle) au moins à la version 10 (la dernière est la 16).
- Il existe aussi des *environnements de développement intégrés* :
  - Eclipse.
  - NetBeans.
  - IntelliJ IDEA.

Structure d'un programme Java

- Un programme Java = une ou plusieurs définitions de classes.
- Une de ces classes au moins doit contenir une méthode de classe `main()` qui sera appelée par la machine virtuelle au lancement du programme.

Prototype de la fonction `main()`

```
public static void main(String[] args);
```

- On notera que le nom de cette méthode commence par une minuscule, comme tous les noms de méthodes en Java.
- Seule une classe « exécutable » a besoin d'une méthode `main()` mais rien n'empêche d'en ajouter une à n'importe quelle classe : cela permet de fournir un exemple d'utilisation et de tester rapidement la classe...

Conventions de nommage et commentaires

- Les noms de classes utilisent le *Pascal Case* (ex : `MaClasse`).
- Les noms des variables et des méthodes utilisent le *Camel Case* (ex : `maMethode`, `uneVariable`).
- Les noms de constantes sont tout en majuscules.
- Les noms des paquetages et les mot-clés du langage sont tout en minuscules.
- Trois types de commentaires :
  - Sur plusieurs lignes, entre `/*` et `*/`.
  - Sur une seule ligne, après `//`.
  - Les lignes de commentaires traités par `javadoc` sont comprises entre `/**` et `*/`.

Système de type

Les types de Java se divisent en :

- *types primitifs* : ils correspondent essentiellement aux types du langage C/C++. Les variables de ces types contiennent directement la valeur.
- *types références* : ils correspondent aux classes. Les objets de ces types contiennent des références vers les véritables instances qui sont allouées sur le tas. Certaines classes sont *immutables* (une fois créés, leurs objets ne peuvent plus être modifiés).
- Il existe une *classe enveloppe* pour chaque type primitif afin d'obtenir un type référence lorsque cela est nécessaire (*boxing* automatique).
- Inversement, les objets de ces classes enveloppes sont convertis en types primitifs lorsque cela est nécessaire (*unboxing* automatique).

Hello world

- Pour simplifier l'affichage et la saisie des programmes en mode console, nous utiliserons les méthodes `nextXXX()` de la classe `Scanner` introduite par Java 1.5.
  - Les méthodes `next()` et `nextLine()` lisent respectivement une chaîne et une ligne.
  - Les méthodes `nextInt()`, `nextFloat()` et `nextDouble()` lisent des nombres (voir la doc de l'API pour plus de détails).
- Pour l'affichage, nous utiliserons les méthodes `printf()`, `print()` et `println()` de `System.out` (ces méthodes affichent des chaînes et appellent la méthode `toString()` de leur paramètre).

Hello World

Hello.java

```
import java.util.Scanner;  (1)

public class Hello {       (2)
  public static void main(String[] args) {  (3)
    Scanner clavier = new Scanner(System.in);
    System.out.print("Bonjour, quel est ton nom ? : ");
    String nom = clavier.nextLine();   // Un nom peut contenir plusieurs mots...
    System.out.print("Enchanté, " + nom + ", quel âge as-tu ? ");
    // ou : System.out.printf("Enchanté, %s, quel âge as-tu ? ", nom);
    int age = clavier.nextInt();
    System.out.println("Tu as " + age + " ans ?");
    // ou : System.out.printf("Tu as %d ans ?%n", age); // Noter le %n...
  }
}
```

1. Ce programme utilise la classe `Scanner`, qui fait partie du paquetage `java.util`. Ce paquetage n'est pas inclus par défaut et il faut donc *importer* la classe.
2. La classe ayant été déclarée `public`, elle doit être stockée dans un fichier portant le même nom que la classe, avec l'extension `.java` (*Hello.java*).
3. La méthode principale doit être déclarée `public` pour pouvoir être appelée de l'extérieur de la classe où elle est définie. En outre, cette méthode est une méthode de classe, ce qui est indiqué par le mot-clé `static`.

Remarque : Depuis Java 10, on peut utiliser le mot-clé `var` pour utiliser l'inférence de type (voir exemple plus loin).

La même chose en Kotlin, Scala et Python

Hello.kt

```
fun main() {
   print("Bonjour, quel est ton nom ? : ")
   val nom = readLine()
   print("Enchanté, $nom, quel âge as-tu ? : ")
   val age = readLine()!!.toInt()
   println("Tu as $age ans ?")
}
```

Hello.scala

```
import scala.io.StdIn

@main def go =
  val nom = StdIn.readLine("Bonjour quel est ton nom ? ")
  val age = StdIn.readLine(s"Enchanté, $nom, quel âge as-tu ? ").toInt
  print(s"Tu as $age ans ?")
```

hello.py

```
nom = input("Bonjour, quel est ton nom ? : ")
age = int(input(f"Enchanté, {nom}, quel âge as-tu ? "))
print(f"Tu as {age} ans ?")
```


Remarques

Classe publique et nom de fichier

- La notion de *publique* n'a de sens pour une classe que lorsqu'elle fait partie d'un paquetage. En ce cas, `public` indique qu'elle est accessible à l'extérieur du paquetage. Elle doit alors être stockée dans un fichier portant le *même nom* qu'elle.
- Une classe non définie dans un paquetage fait partie d'un *paquetage par défaut* et sera toujours visible : elle n'a donc pas besoin d'être déclarée publique et peut être stockée dans un fichier de nom quelconque.
- Dans les deux cas, le nom du fichier `.class` obtenu par compilation sera celui de la classe, pas du fichier.

Contenu du fichier Bla.java

```
class Hello {
  public static void main(String[] args) {
    System.out.println("Hello");
  }
}
```

Compilation et exécution de la classe Hello

```
% javac Bla.java
% java Hello
Hello
```

Remarques

`nextInt()` et exceptions

- La méthode `nextInt()` s'attend à lire un entier (ou, plutôt, une suite de caractères susceptible de représenter un entier)...
- Dans le cas contraire, elle lève l'exception `InputMismatchException`.
- Les appels aux méthodes `nextXXX()` doivent donc toujours avoir lieu dans un bloc `try` (voir la section sur les exceptions).

Utilisation correcte de `nextInt()`

```
import java.util.Scanner;
import java.util.InputMismatchException;

public class Hello {
  public static void main(String[] args) {
    var clavier = new Scanner(System.in);
    System.out.print("Bonjour, quel est ton nom ? : ");
    var nom = clavier.nextLine();
    var age = 0;
    do {  // On boucle tant qu'on n'a pas un age correct...
      System.out.print("Enchanté, " + nom + ", quel âge as-tu ? ");
      try {
       age = clavier.nextInt();
      } catch(InputMismatchException e) {
       clavier.next(); // pour sauter la mauvaise saisie
      }
    } while (age <= 0);
    System.out.println("Tu as " + age + " ans ?");
  }
}
```


Types primitifs

Les variables et constantes de ces types stockent directement leurs valeurs, comme en C. Ce ne sont donc pas des *objets* au sens POO du terme...

Type	Enveloppe	Taille
`boolean`	`Boolean`	1 bit
`char`	`Character`	16 bits
`byte`	`Byte`	8 bits
`short`	`Short`	16 bits
`int`	`Integer`	32 bits
`long`	`Long`	64 bits
`float`	`Float`	32 bits
`double`	`Double`	64 bits

Types primitifs

- Tous les nombres sont signés. Le type `char` est compatible avec les entiers.
- Les représentations de tous ces types sont indépendantes de la plateforme (contrairement à C/C++).
- Les classes enveloppes (immutables) dérivent de la classe `Object`. Elles permettent de manipuler les types primitifs comme des objets et fournissent des constantes, des méthodes utilitaires et de conversion. Le *boxing*/*unboxing* est automatique.
- Les calculs sur les entiers utilisent `int` et `long` : les variables des autres types entiers sont automatiquement converties en `int` avant que les calculs n'aient lieu.
- Les classes enveloppes des nombres fournissent les constantes `MIN_VALUE` et `MAX_VALUE`.
- Toutes les variables sont initialisées à 0 par défaut (à `false` pour les booléens).

Littéraux des types primitifs

- Les deux seuls littéraux booléens sont `false` et `true`.
- Un littéral entier est, par défaut, considéré comme un `int`. Pour indiquer qu'il s'agit d'un long on utilise le suffixe `L`.
- Un littéral entier est, par défaut, considéré comme du décimal. Avec le préfixe `0x` ou `0X` il est considéré comme de l'hexadécimal ; avec le préfixe `0b` ou `0B`, il est considéré comme du binaire.
- Un littéral réel est, par défaut, un `double`. Pour indiquer qu'il s'agit d'un `float` on utilise le suffixe `f` ou `F`.
- Les littéraux réels peuvent s'exprimer en virgule flottante ou en notation « mantisse et exposant ».
- On peut séparer des groupes de chiffres par le caractère `_` pour améliorer la lisibilité des grands nombres.
- Les littéraux caractères s'écrivent comme un caractère entre deux apostrophes simples, ou comme un littéral entier sur 16 bits.

Les classes

- Les classes de Java sont définies dans l'API, qui est subdivisée en paquetages. Le paquetage `java.lang` est automatiquement connu par tout programme Java et contient les classes essentielles comme `String`, `StringBuilder`, `System` et `Math`. Les autres paquetages doivent être importés explicitement à l'aide de l'instruction `import`.
- L'ensemble des classes Java forme une arborescence d'héritage dont la racine est la classe `Object` : toutes les classes disposent donc des méthodes de `Object` et toute instance d'une classe quelconque est une instance d'`Object`. On peut donc utiliser une instance de classe partout où un `Object` est attendu.
- Un objet d'une classe est décrit par une *référence* et doit être instancié explicitement par un appel au constructeur de la classe (la classe `String`, les tableaux et les énumérations disposent d'une syntaxe simplifiée).

Les classes

Toutes les classes de Java dérivant de la classe `Object`, ses méthodes sont donc utilisables par tous les objets et certaines peuvent être redéfinies (`toString()`, `equals()`, `hashcode()` et `clone()` notamment).

- `public boolean equals(Object obj)` teste l'égalité du contenu de `this` et de `obj`. Par défaut, cette méthode ne renvoie `true` que si les deux références désignent le même objet (`this` == `obj`).
- `public int hashcode()` renvoie un code de hachage pour l'objet.
- `public String toString()` renvoie une représentation d'un objet sous forme de chaîne. Les sous-classes peuvent redéfinir cette méthode, qui est automatiquement utilisée par la concaténation et les méthodes d'affichage.
- `protected Object clone()` effectue et renvoie une copie binaire de l'objet. Voir la seconde partie du cours. Cette méthode étant `protected`, il faut la redéfinir comme `public` pour en disposer.