

# Java CM1

Olivier Marchetti

Laboratoire d'informatique de Paris 6 – Pôle SoC – Sorbonne Université

17 septembre 2021



# Module JAVA 2021-2022

## Organisation :

- ▶ 14 CM-TD de 2h,
- ▶ 8 TP de 4h.

## Modalités d'évaluation :

- ▶ 2 TP en mode projet<sup>a</sup> :
  - dont 1h de questions de cours sur papier (sans document),
  - et 3h de TP « *brainstorming* ».
- ▶ éventuelle bonification de la moyenne selon la qualité de certaines préparations de TP.

---

<sup>a</sup> selon l'évolution sanitaire.

## Conseils :

- ▶ travail régulier du cours (donc **apprendre** son cours),
- ▶ reprendre les exemples du cours (pour apprendre rapidement la syntaxe et gagner en vitesse),
- ▶ finir les TP soi-même.

Bibliographie : trop vaste... allez faire un tour en BU.

## Animation du module :

- ▶ CM-TD : moi-même.
- ▶ TP :
  - Célia Mahamdi,
  - Moi-même.

# Module JAVA 2020-2021 : mode d'emploi

- ▶ Chaque transparent du support électronique est muni de boutons :



Ce logo en haut à gauche permet d'atteindre la table des matières. Les entrées à droite du logo sont accessibles par simple clic.



La pagination permet la navigation et le « rembobinage » d'une éventuelle animation.

- ▶ Ce cours est accompagné d'un certain nombre de codes sources. Pour encourager/faciliter l'apprentissage, certains sont munis d'un logo :



Ce logo vous invite à retaper le code source afin de vous familiariser avec la syntaxe du langage.



Ce logo vous permet, après un ou deux clics de souris, d'éditer le code source<sup>b</sup>, de réaliser des tests.

- ▶ Le langage JAVA est doté de concepts et d'outils. Son compilateur produit des messages utilisant un vocabulaire précis.

⇒ il faut maîtriser son cours !

b. Ce code source sera conforme à la convention d'écriture de JAVA (à savoir avec des tabulations de largeur huit, contrairement à la largeur quatre utilisée pour ce support).

# Plan

- 1 Aperçu & Historique
- 2 Types primitifs, opérateurs, conversions, constantes
  - Types primitifs
  - Opérateurs
  - Conversions de types
  - Constantes en JAVA
- 3 Tableaux et chaînes de caractères
  - Tableaux
  - Chaînes de caractères
- 4 Structures de contrôle
- 5 Classes, champs et méthodes
- 6 Concepts de la programmation orientée objet
  - Modélisation
  - Encapsulation des données
- 7 Bonnes pratiques et outils
  - Bonnes pratiques
  - Outils pour la POO et JAVA



- 1 Aperçu & Historique
- 2 Types primitifs, operateurs, conversions, constantes
  - Types primitifs
  - Opérateurs
  - Conversions de types
  - Constantes en JAVA
- 3 Tableaux et chaînes de caractères
  - Tableaux
  - Chaînes de caractères
- 4 Structures de contrôle
- 5 Classes, champs et méthodes
- 6 Concepts de la programmation orientée objet
  - Modélisation
  - Encapsulation des données
- 7 Bonnes pratiques et outils
  - Bonnes pratiques
  - Outils pour la POO et JAVA

# JAVA en quelques mots – à ses débuts

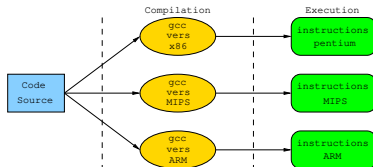
- ▶ Java est un langage créé par la société SUN en 1995 (partant d'un projet initié en 1991).
  - Appartient à la famille des langages impératifs.
  - Langage interprété.
- ▶ Les objectifs de ce langage sont :
  - ❶ Langage respectant le modèle objet.
  - ❷ Syntaxe proche du C/C++.
  - ❸ Efficacité.
  - ❹ Pouvant tourner sur toutes les machines possibles (quelque soit le système d'exploitation, quelque soit l'architecture).
  - ❺ Orienté vers l'internet.

## Remarque

Bien qu'étant un langage orienté pour l'internet, JAVA n'a rien à voir avec le langage JAVASCRIPT.

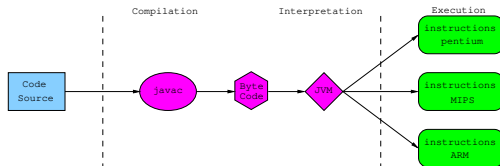
# JAVA en quelques mots – force principale

- ▶ L'atout principal de JAVA est sa portabilité absolue.
- ▶ Le *leitmotiv* de JAVA est *"Write Once, Run Anywhere"*.
  - Rappel : langage compilé avec GCC (comme le C)



✗ Régler/recompiler pour chaque architecture (ou OS) cible.

- Portabilité de JAVA : langage interprété (et compilé avec javac)



✓ Une seule compilation !

# JAVA en quelques mots – *pro et contra*

## ► *Pro*

- portabilité absolue (de la cafetière jusqu'au supercalculateur),
- syntaxe similaire au C/C++ (facilité d'apprentissage),
- sécurité de la JVM,
- gestion des erreurs,
- gestion automatique de la mémoire (principe du ramasse-miettes),
- Librairie officielle particulièrement fournie et documentée.

## ► *Contra*

- Langage interprété (performances limitées).

### Améliorer les performances de l'interprétation : la technique JIT

Les portions de *bytecode* fréquemment interprétées sont traduites et stockées par la JVM.

⇒ amélioration des performances

C'est la technique de compilation dite « juste-à-temps » (*just in time*).

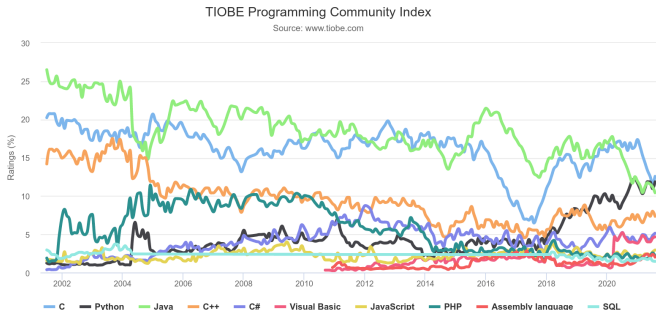


# JAVA en quelques mots – aujourd'hui

Malgré

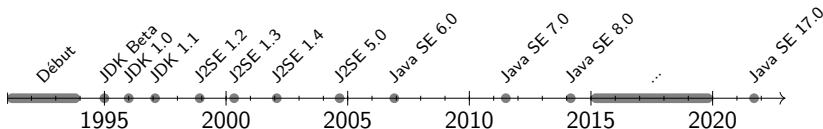
- ▶ les aléas de la société SUN (rachetée par la société ORACLE en 2010... après des pressions secrètes des USA sur l'UE !),
- ▶ une crise de confiance au niveau sécurité (tournant 2010),
- ▶ des performances moindre que d'autres langages (*i.e.* C/C++),

le langage JAVA reste et demeure un langage très utilisé/populaire dans le monde (e.g. enquête TIOBE).



# JAVA en quelques mots – aujourd'hui et demain

JAVA a connu depuis sa création de nombreuses évolutions qui font de ce langage un langage dit multi-paradigmes :



Quelques jalons :

- ▶ 1998 – J2SE 1.2 : composants graphiques SWING,
- ▶ 2007 – OpenJDK : première version libre,
- ▶ 2008 – J2SE 5.0 : types génériques,
- ▶ 2014 – Java SE 8.0 : couche fonctionnelle,
- ▶ 2021 – Java SE 17.0 : API de plus de 4800 classes.

Aujourd'hui : une nouvelle jeunesse grâce à ANDROID (utilise une machine virtuelle spécifique DALVIK ou ANDROID RUNTIME – ART).

Les *bytecodes* sont spécifiques et... incompatibles !

# Premiers pas avec JAVA

- ▶ Pour programmer en JAVA, il faut disposer du fameux JDK disponible pour :

- toutes les distributions GNU-LINUX.

Par exemple pour les distributions de UBUNTU ou DEBIAN, on installera :

OpenJDK Java 8

- WINDOWS,
- MACOS,
- SOLARIS.

Le JDK propose notamment :


- un chargeur de programme (java),
- un compilateur (javac),
- un débogueur (jdb),
- un outil de documentation (javadoc).

- ▶ Pour seulement faire tourner des programmes JAVA, il faut disposer de la JVM (le fameux JRE – *Java Runtime Environment*).

# Premier programme en JAVA

Pour éditer du code source JAVA, on utilise un simple éditeur de texte.

```
1 public class HelloWorld {  
2     // Ceci est un commentaire.  
3     public static void main(String args[]) {  
4         System.out.println("Hello world !");  
5     }  
6 }
```



Le terminal affichera la chaîne de caractères "Hello world !".

Remarques :

- ▶ au moins une classe dans le fichier (ici HelloWorld.java),
- ▶ les instructions sont toutes localisées au sein des classes,
- ▶ une méthode<sup>1</sup> main() pour piloter le programme.

---

<sup>1</sup> En JAVA, on parle de méthodes et non de fonctions.

# Compilation sous LINUX

- ▶ Commande de compilation du fichier source avec javac :

```
[22:05] [Prog pc666 :]$ ls
HelloWorld.java
[22:05] [Prog pc666 :]$ javac HelloWorld.java
[22:05] [Prog pc666 :]$ ls
HelloWorld.class HelloWorld.java
```

- Cette commande génère un fichier *bytecode* HelloWorld.class.
- Ce fichier *bytecode* est une représentation intermédiaire du programme destinée à être interprétée par la machine virtuelle JAVA.

- ▶ Lancement du programme avec la commande java :

```
[22:08] [Prog pc666 :]$ java HelloWorld
Hello world !
```

La JVM exécute la méthode `main()` de la classe HelloWorld.

# La méthode main()

- ▶ Une classe contient au plus une méthode `main()`.
- ▶ Cette méthode est le point d'entrée du programme à l'exécution.
- ▶ Soit un fichier `mainAouB.java` contenant deux classes A et B :

```

1 class A {
2     public static void main(String args[]) {
3         System.out.println("Méthode main() de la classe A");
4     }
5 }
6
7 class B {
8     public static void main(String args[]) {
9         System.out.println("Méthode main() de la classe B");
10    }
11 }

```

La compilation génère autant de fichiers *bytecode* qu'il n'y a de classes dans le fichier source compilé.

```

[14:05][Prog pc666 :]$ javac mainAouB.java
[14:05][Prog pc666 :]$ ls
A.class    B.class    mainOuB.java

```

La méthode `main()` utilisée par la JVM sera celle figurant dans la classe invoquée sur la ligne de commande :

```

[14:05][Prog pc666 :]$ java A
Méthode main() de la classe A
[14:05][Prog pc666 :]$ java B
Méthode main() de la classe B

```

- ▶ Le programmeur peut y laisser des instructions illustrant des tests.

# Affichage dans un terminal avec JAVA

Tout programme JAVA possède un flux de sortie standard noté `System.out`<sup>1</sup>, positionné par défaut au terminal.

```
class SortieTerminal {  
    public static void main (String args[]) {  
        // Sortie JAVA.  
        System.out.println("Le système solaire comporte");  
        int nbPlaneteTellurique = 4;  
        int nbPlaneteGazeuse = 4;  
        System.out.print("_ " + nbPlaneteTellurique + " planètes telluriques,\n");  
        System.out.println("_ " + nbPlaneteGazeuse + " planètes gazeuses.");  
  
        // Sortie JAVA façon langage C.  
        String message = "Pluton n'est plus une planète !";  
        System.out.printf("%s\n", message);  
    }  
}
```

► Méthodes sans aucun code format :

- `System.out.print()` met en forme et concatène les données fournies,
- `System.out.println()` idem + produit un retour à la ligne automatiquement.

► Méthode avec code format (pénible à gérer) : `printf()`.

---


<sup>1</sup> `System.out` est aussi le flux de sortie standard d'erreur.

# Lecture au clavier avec JAVA

Tout programme JAVA possède un flux d'entrée standard noté `System.in`, positionné par défaut au clavier. On utilise un objet `Scanner`.

```
import java.util.Scanner;

class EntreeTerminal {
    public static void main(String args[]) {
        Scanner clavier = new Scanner(System.in);
        System.out.println("Quel âge a le système solaire (en Ma) ?");
        double age = clavier.nextDouble();
        System.out.println("Où se trouve la ceinture d'astéroïdes ?");
        clavier.nextLine(); // Bogue si retrait.
        String reponse = clavier.nextLine();
        System.out.println("Vos réponses : " + age + " " + reponse);
    }
}
```



- ▶ La JVM ne peut pas deviner le type des données saisies.
  - méthode pour lire un `int`,
  - méthode pour lire un `double`,
  - méthode pour lire une `String`...

## Les entrées/sorties en JAVA

Les entrées/sorties sont délicates et auront un cours complètement dédié.



# La documentation – une pratique essentielle

Pour documenter son code, on utilisera les balises suivantes

```
// mise en commentaire du reste de la ligne.  
  
/*  
    mise en commentaire de plusieurs lignes.  
*/
```

## Du mésusage des commentaires

Il existe deux excès :

- ❶ ne jamais commenter,  
⇒ le code doit durer !
- ❷ tout commenter,  
⇒ Le code doit être lu facilement par autrui !

## Du bon usage des commentaires

Il faut commenter au fur-et-à-mesure :

- ❶ toutes les méthodes au niveau de l'entête.
- ❷ certains champs si nécessaire,
- ❸ certains passages délicats.

- 1 Aperçu & Historique
- 2 Types primitifs, opérateurs, conversions, constantes
  - Types primitifs
  - Opérateurs
  - Conversions de types
  - Constantes en JAVA
- 3 Tableaux et chaînes de caractères
  - Tableaux
  - Chaînes de caractères
- 4 Structures de contrôle
- 5 Classes, champs et méthodes
- 6 Concepts de la programmation orientée objet
  - Modélisation
  - Encapsulation des données
- 7 Bonnes pratiques et outils
  - Bonnes pratiques
  - Outils pour la POO et JAVA

# Les types primitifs de JAVA – liste

Ce sont les types de base pour stocker des informations simples de nature :

► numériques

Entiers	Nombre d'octets
byte	1
short	2
int	4
long	8

Flottants	Nombre d'octets
float	4
double	8

► textuelles

Caractères	Nombre d'octets
char	2

► booléennes

Booléen	Nombre d'octets
boolean	JVM dépendant

Valeurs possibles true/false.

# Les types primitifs de JAVA – types entiers

- ▶ En JAVA, tous les types entiers sont signés.
- ▶ On mettra un "L" (ou un "l") pour que javac interprète correctement cette constante.

## ▶ Exemples :

```
byte nbCouleurs = 16;  
short nbElevesTricheurs = 2;  
int populationEU = 520000000;  
long distNeptuneSoleil = 4500000000L;
```

### Focus sur l'encodage des entiers en JAVA : le complément à 2

Si  $n$  est le nombre de bits d'un entier, sa notation et sa valeur s'écrivent :

$$(a_{n-1}a_{n-2}\dots a_1a_0)_{\overline{2}} = -a_{n-1} \cdot 2^{n-1} + a_{n-2} \cdot 2^{n-2} + \dots + a_1 \cdot 2^1 + a_0 \cdot 2^0$$

- ▶ Signe codé sur le bit de poids fort avec :
  - $a_{n-1} = 0$  si l'entier représenté est positif,
  - $a_{n-1} = 1$  si l'entier représenté est négatif, ajouter 1 au module.
- ▶ Module encodé en binaire ou en binaire complémenté.

# Les types primitifs de JAVA – types flottants

- ▶ En informatique, on ne manipule qu'un nombre fini d'approximations de réels.

✗ Problèmes de fiabilité...

- ▶ Exemples :

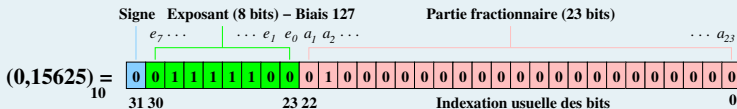
```
float moyenne = 10.84f; /* le f indique
                          que l'on veut
                          une précision
                          simple. */
double ecartType = 1.751;
```

## Focus sur l'encodage des flottants : les formats IEEE

En base  $b$ , un réel  $x$  peut s'écrire en notation scientifique normalisée  $x = \epsilon \cdot b^e \cdot m$  avec

- ▶  $\epsilon \in \{-1, 1\}$  codant le signe.
- ▶  $m = a_0, a_1 a_2 \dots a_i \dots \in [1, b[$  codant la mantisse (i.e.  $a_0 \neq 0$ ).
- ▶  $e = e_k \dots e_0 \in \mathbb{Z}$  codant l'exposant.

Les formats binaires IEEE encodent le signe  $\epsilon$ , puis un exposant « biaisé »  $e$ , puis la partie fractionnaire de la mantisse  $m$ . Pour un flottant simple précision on aura :



# Types primitifs & représentation

Soient :

- ▶  $\mathbb{B}$  l'ensemble des valeurs prises par un `byte` ;
- ▶  $\mathbb{S}$  l'ensemble des valeurs prises par un `short` ;
- ▶  $\mathbb{I}$  l'ensemble des valeurs prises par un `int` ;
- ▶  $\mathbb{L}$  l'ensemble des valeurs prises par un `long`.

Alors nous avons :

$$\mathbb{B} \subset \mathbb{S} \subset \mathbb{I} \subset \mathbb{L}$$

De même, si l'on désigne par :

- ▶  $\mathbb{F}$  l'ensemble des valeurs prises par un `float` ;
- ▶  $\mathbb{D}$  l'ensemble des valeurs prises par un `double`<sup>1</sup> ;

alors :

$$\mathbb{F} \subset \mathbb{D}$$

Attention car les relations inverses ne sont pas vraies !

<sup>1</sup> Les tailles des champs pour l'exposant et la partie fractionnaire sont 11 et 52 bits.

# Les types primitifs de JAVA - type textuel : UTF8

Les caractères sont encodés en binaire à l'aide d'une table d'encodage par de petits entiers positifs.

Table d'encodage ASCII :

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

- ▶ Le caractère '0' est encodé par  $3 \times 16 = 48$  (soit 0x30), le caractère '1' par 49...
- ▶ Le caractère 'A' est encodé par  $4 \times 16 + 1 = 65$  (soit 0x41), le caractère 'B' par 66...
- ▶ Le caractère 'a' est encodé par  $6 \times 16 + 1 = 97$  (soit 0x61), le caractère 'b' par 98...

# Les types primitifs de JAVA - type textuel

```
class AfficherCaractere {
    public static void main(String args[]) {
        char c1 = 'E', c2 = 'I', c3 = '-', c4 = '2', c5 = 'I';
        System.out.println(c1 + c2 + c3 + c4 + c5);
        System.out.println(" " + c1 + c2 + c3 + c4 + c5);
        System.out.println(c1 + c2 + c3 + c4 + c5 + " ");
    }
}
```



```
[09:03][Prog pc666 :]$ java AfficherCaractere
310
EI-2I
310
```

- ▶ Le premier affichage n'est que la somme des variables.
- ▶ **Attention** : 'x' ≠ "x".

## Codage des caractères – rappel

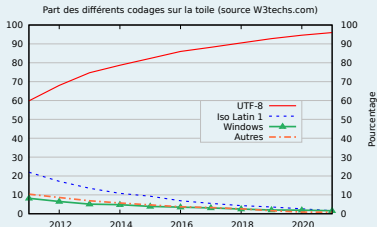
Ils sont codés par des entiers avec une convention. JAVA utilise UNICODE.

Convention	Nombre de bits	Nombre de symboles*
ASCII	7	128
ISO-LATIN 1	8	256
UNICODE – UTF-8	16	65536

HTML5 utilise UTF-8.

\* : ou caractères de contrôle.

UTF8 est rétrocompatible, c'est-à-dire elle contient ASCII.





# L'opérateur de conversion de types en JAVA – cast ()

- ▶ En JAVA, on peut « convertir » une valeur d'un type vers un autre en utilisant l'opérateur unaire dit de cast de syntaxe :

```
type resultat = (type) (expression);
```

- On parle aussi de transtypage explicite.
  - Conversions possibles entre les types numériques et le type textuel.
  - Conversions impossibles depuis et vers le type boolean, les types tableaux.
- ▶ Il faut prendre soin que la valeur « convertie » soit correcte !

```
short nbElevesTricheurs = 3;  
long populationUE = 520000000;  
nbElevesTricheur = (short) populationUE;      X FAUX !
```

- Le compilateur ferme le yeux et laisse au programmeur la responsabilité de ses conversions.

# Retour sur le type textuel – 1/2

## Entiers, caractères & table d'encodage

- ▶ Les lettres et chiffres ont le bon goût de se suivre.
- ▶ Les types textuels étant *in fine* des entiers, il est possible de faire des calculs très pratiques !

Première tentative d'affichage d'un caractère et de son code :

```
1 public class Lettre {
2     public static void main(String args[]) {
3         char premiereLettre = 'A';
4         char lettreMystere = premiereLettre + 25;
5         System.out.printf("Code de %c = %d\n",
6                             lettreMystere,
7                             lettreMystere);
8     }
9 }
```

```
[15:22] [Prog pc666 :]$ javac Lettre.java
Lettre.java:4: error: incompatible types: possible
    lossy conversion from int to char
        char lettreMystere = premiereLettre + 25;
                                           ^
1 error
```

- ▶ Le compilateur javac est peu permissif.
- ▶ Aucun *bytecode* n'est généré.

⇒ corriger la ligne 4 et effectuer un cast().

# Retour sur le type textuel – 2/2

Deuxième tentative d'affichage d'un caractère et de son code :

```
1 public class Lettre {  
2     public static void main(String args[]) {  
3         char premiereLettre = 'A';  
4         char lettreMystere = (char) (premiereLettre + 25);  
5         System.out.printf("Code de %c = %d\n",  
6                             lettreMystere,  
7                             lettreMystere);  
8     }  
9 }
```



Malgré une compilation réussie, le programme échoue :

```
[15:28] [Prog pc666 :]$ javac Lettre.java  
[15:28] [Prog pc666 :]$ java Lettre  
Code de Z =  
Exception in thread "main" java.util.IllegalFormatConversionException: d != java.lang.Character  
    at java.util.Formatter$FormatSpecifier.failConversion(Formatter.java:4302)  
    ...  
    at Lettre.main(Lettre.java:5)  
[15:28] [Prog pc666 :]$
```

Une « exception »<sup>1</sup> est générée à l'exécution et provoque un arrêt brutal.

⇒ corriger la ligne 5 indiquée par l'exception en mettant un cast() devant le dernier argument.

<sup>1</sup> Le mécanisme des exceptions sera étudié lors du CM3.

# Les opérateurs en JAVA – Binaires

On retrouve peu ou prou les mêmes opérateurs qu'en C :

- ▶ affectation : `=`
- ▶ arithmétique : `+`, `-`, `*`, `/`, `%`, `++`, `--`, `+=`, `-=`, `*=`, `/=`, `%=`
- ▶ comparaison : `<`, `>`, `<=`, `>=`, `==`, `!=`
- ▶ logique : `!`, `|`, `&`, `^`
  - avec interruption dès que la valeur peut-être déterminé `&&` et `||`.
- ▶ manipulation des motifs binaires (Électronique/TdS/Image) :  
`~`, `>>`, `<<`, `&`, `|`, `^`, `>>=`, `<<=`, `&=`, `|=`, `^=`

## Lisibilité des expressions

La JVM dispose de règles d'évaluation précises pour les expressions formées avec ces opérateurs...

- ▶ Inutile d'apprendre ces règles !
- ▶ Préférer un bon parenthésage.
- ▶ Disposer un espace avant et après TOUS les opérateurs binaires !

# Les opérateurs en JAVA – Ternaire ' ? '

Il arrive souvent que le programmeur écrive :

```
if (conditionTest) {  
    blocInstructionsVRAI;  
}  
else {  
    blocInstructionsFAUX;  
}
```

Lorsque :

- ▶ la condition du test est simple,
- ▶ chacun des deux blocs est composé d'une seule instruction,

il peut être préférable d'utiliser l'opérateur ternaire pour gagner en lisibilité (code plus court) :

```
conditionTest ? instructionVRAI : instructionFAUX;
```

## Rappel utile

L'objectif de tout programmeur est de produire un code lisible !

# Les opérateurs en JAVA – Concaténation '+'

## Observation générale

Beaucoup de programmes manipulent des chaînes de caractères.

⇒ JAVA prévoit un opérateur de concaténation.

```
1 import java.util.Calendar;
2
3 class Concatenation {
4     public static void main (String args[]) {
5         String debMessage = "JAVA est un langage très populaire";
6         String finMessage = " avec une API très riche.";
7         String messageComplet = debMessage + finMessage;
8         System.out.println(messageComplet);
9         int ageJava = Calendar.getInstance().get(Calendar.YEAR) - 1991;
10        String phrase = "JAVA a " + ageJava + " ans";
11        System.out.println(phrase);
12    }
13 }
```



```
[10:29][Prog pc666 :]$ java Concatenation
JAVA est un langage très populaire avec une API très riche.
JAVA a 30 ans.
```

## Remarque

Le mot clé import n'importe rien du tout ! Il sera étudié lors du CM2.

# Les opérateurs en JAVA – Concaténation '+'

```
1 import java.util.Calendar;
2
3 class Concatenation {
4     public static void main (String args[]) {
5         String debMessage = "JAVA est un langage très populaire";
6         String finMessage = " avec une API très riche.";
7         String messageComplet = debMessage + finMessage;
8         System.out.println(messageComplet);
9         int ageJava = Calendar.getInstance().get(Calendar.YEAR) - 1991;
10        String phrase = "JAVA a " + ageJava + " ans";
11        System.out.println(phrase);
12    }
13 }
```

À l'exécution :

```
[10:29][Prog pc666 :]$ java Concatenation
JAVA est un langage très populaire avec une API très riche.
JAVA a 30 ans.
```

On distingue ici deux types de concaténation :

- ▶ Ligne 7 : réalisée au moment de la compilation (statiquement).
- ▶ Ligne 10 : réalisée au moment de l'exécution (dynamiquement).

# Types primitifs & représentation : conversions implicites

Pour effectuer le calcul d'une expression mixte<sup>1</sup>, la JVM utilise la hiérarchie :

`int` → `long` → `float` → `double`

Ainsi les valeurs numériques sont ajustées au moment du calcul :

```
int nombreChelou = 1664;  
long nombreMarseillais = 51;  
float nombrePasNet = 8.6f;  
nombreChelou + nombreMarseillais + nombrePasNet;
```

Lors du calcul, `nombreChelou` est converti en `long`, puis le résultat de `nombreChelou + nombreMarseillais` est converti en `float`.

Les conversions :

- ▶ `int` → `long` ou `float` → `double` sont sans perte d'information ;
- ▶ d'entiers vers flottants peuvent se faire avec perte d'information (mais préservent l'ordre de grandeur).

---

<sup>1</sup> i.e. une mettant en jeu des opérandes de types différents.



# Promotion numérique

Lorsqu'une expression utilise des types tels que `byte`, `short` ou `char`, alors les valeurs sont « promues » en type `int` d'abord.

`byte` → `int`  
`short` → `int`  
`char` → `int`

```
byte nbElevesEI2I = 120;  
short nbElevesPolytechSorbonne = 1234;  
short nbElevesHorsEI2I = (short) (nbElevesPolytechSorbonne - nbElevesEI2I);
```

À l'exécution, la JVM :

- ▶ convertira en `int` les deux opérandes de la soustraction ;
- ▶ effectuera le calcul (donc un résultat en `int`) ;
- ▶ convertira de force ce résultat en `short` (perte possible d'information).


## Exemple d'utilisation

On pourra alors se servir de ces types pour économiser de la mémoire.

# Les constantes en JAVA – le mot clé `final`

Une constante sera précédée du mot clé `final`, par exemple :

```
1 class ExempleConstanteJava {
2     public static void main (String [] args) {
3         final int VITESSE_LUMIERE_VIDE = 299792458; // en m/s.
4         final double UNITE_ASTRONOMIQUE = 149597870.7; // en km, -distance Soleil/Terre.
5         int nbSecondeTrajetLumiere = (int) (UNITE_ASTRONOMIQUE * 1000 / VITESSE_LUMIERE_VIDE);
6         int nbMinuteTrajetLumiere = nbSecondeTrajetLumiere / 60;
7         int reste = nbSecondeTrajetLumiere % 60;
8         System.out.println("Depuis le soleil, la lumière met " + nbMinuteTrajetLumiere + " minutes et " +
9             reste + " secondes pour nous parvenir." );
10    }
11 }
12 /*
13  La distance Terre-Mars pouvant varier entre 50 et 400 millions de km, en déduire que...
14  */
```



## Remarques

- ▶ Une constante déclarée en `final` peut-être aussi initialisée à l'exécution.
- ▶ En JAVA, il n'existe pas de constantes symboliques comme en C.
- ▶ Les identifiants de ces variables sont entièrement en majuscules.

# Les constantes en JAVA – notion d'expression constante

- C'est une expression dont la valeur est calculable à la compilation.

```

1  class ExpressionConstante {
2      public static void main (String args[]) {
3          double PI = 3.1415;
4          double nbEuler = 2.7182;
5          double somme = 3.1415 + 2.7182;
6          somme = PI + nbEuler;
7
8          final double cstePI = 3.1415;
9          final double csteNbEuler = 2.7182;
10         somme = cstePI + csteNbEuler;
11     }
12 }

```

- Ligne 3 à 5 : les membres droits sont des expressions constantes.
- Ligne 6 : le membre droit n'est pas une expression constante.
- Ligne 10 : l'instruction entière est une expression constante.

- Certaines expressions bien déclarées en `final` ne sont pas des expressions constantes.

```

1  import java.util.Scanner;
2
3  class FinalVsExpressionConstante {
4      public static void main (String args[]) {
5          final int nbLimite;
6          Scanner clavier = new Scanner(System.in);
7          nbLimite = clavier.nextInt();
8          System.out.println(nbLimite);
9      }
10 }

```

- Une variable déclarée en `final` sera affectée au plus une fois, éventuellement de manière différée (cf. exemple).
- Ici, la valeur est déterminée seulement à l'exécution.

- 1 Aperçu & Historique
- 2 Types primitifs, operateurs, conversions, constantes
  - Types primitifs
  - Opérateurs
  - Conversions de types
  - Constantes en JAVA
- 3 Tableaux et chaînes de caractères
  - Tableaux
  - Chaînes de caractères
- 4 Structures de contrôle
- 5 Classes, champs et méthodes
- 6 Concepts de la programmation orientée objet
  - Modélisation
  - Encapsulation des données
- 7 Bonnes pratiques et outils
  - Bonnes pratiques
  - Outils pour la POO et JAVA

# Les tableaux

Un tableau est un ensemble d'éléments indexés de même type.

► Deux syntaxes pour déclarer un tableau :

- syntaxe ordinaire :

```
type unTableau[];  
type autreTableau[], variable;  
// variable n'est donc pas un tableau.
```

- syntaxe multidéclaration :

```
type [] unTableau, autreTableau;
```

► Deux syntaxes pour définir et donc allouer en mémoire un tableau :

- Déclaration et définition simultanée :

```
type unTableau[] = {expr1, expr2, ..., exprN};
```

On pourra préciser entre crochets le nombre d'éléments voulus.

- Utilisation de l'opérateur new avec la syntaxe :

```
unTableau = new type[nbElementsVoulus];
```

Les éléments sont « mis à zéro ».

# Les tableaux – accès aux éléments

En informatique,

- ▶ les tableaux sont indexés à partir de 0 et **non de 1**,
- ▶ les accès aux éléments d'un tableau se font en temps constant (*i.e.* accéder au premier ou au dernier élément « ne coûte pas plus cher »).

La syntaxe :

```
tableau[indiceElement]
```

désigne la case indexée par indiceElement.

```
1 class ExempleTableau {
2     public static void main (String args[]) {
3         int tabNote[] = {6, 2, 13, 15, 10, 9, 17, 17, 9, 19};
4         int sommeNote = 0;
5         float moyenne = 0.0f;
6         tabNote[6] = tabNote[7] = 0; // Tricheurs.
7         for (int i = 0; i < 10; i++) {
8             sommeNote += tabNote[i];
9         }
10        moyenne = sommeNote / 10;
11        System.out.println("Moyenne de la promotion : " + moyenne);
12    }
13 }
```

Est-ce bien le bon résultat ?

```
[18:44] [Prog pc666 :] $ java ExempleTableau
Moyenne de la promotion : 8.0
```

Remarque :

double affectation.

# Les tableaux – quelques spécificités

Le résultat est incorrect.

```
class ExempleTableau {
    public static void main (String args[]) {
        int tabNote[] = {6, 2, 13, 15, 10, 9, 17, 17, 9, 19};
        int sommeNote = 0;
        float moyenne = 0.0f;
        tabNote[6] = 0; // Tricheur.
        tabNote[7] = 0; // Tricheur.
        for (int i = 0; i < 10; i++) {
            sommeNote += tabNote[i];
        }
        moyenne = sommeNote / 10.0f;
        System.out.println("Moyenne de la promotion : " + moyenne);
    }
}
```

● Style : préférer une affectation par ligne.

● Attention : la division sur des opérandes entières est la **division euclidienne**.

Style toujours : utiliser la propriété `length` du tableau :

```
...
for (int i = 0; i < tabNote.length; i++) {
    sommeNote += tabNote[i];
}
moyenne = ((float) sommeNote) / tabNote.length;
...
```

# Les tableaux – spécificités

En JAVA, un identifiant de tableau peut-être réaffecté pour désigner un autre tableau (*i.e.* c'est un pointeur ordinaire).

```

1 class Tableau {
2     public static void main(String args[]) {
3         int tab[] = {6, 2, 13};
4         int autreTab[] = new int[5];
5         System.out.println("Contenu de autreTab :");
6         for (int i = 0; i < autreTab.length; i++) {
7             System.out.print(autreTab[i] + " ");
8         }
9         System.out.println("\nContenu de autreTab :");
10        autreTab = tab;
11        for (int i = 0; i < autreTab.length; i++) {
12            System.out.print(autreTab[i] + " ");
13        }
14        System.out.println();
15    }
16 }

```

```

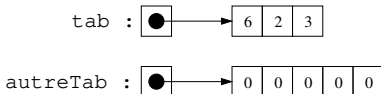
[10:20][Prog pc666 :]$ java Tableau
Contenu de autreTab :
0 0 0 0 0
Contenu de autreTab :
6 2 13

```

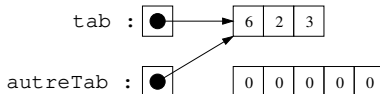
Remarques :

- usage de print et non println.
- réaffectation du pointeur d'un tableau.

- En mémoire, avant exécution de l'instruction de la ligne 10 :



- Après exécution de l'instruction de la ligne 10 :





# Les chaînes de caractères

Représentation des chaînes de caractères avec deux types non-primitifs :

- **String** : afin de stocker une « constante chaîne ».

```
class ExempleString {
    public static void main (String args[]) {
        String message = "Hello World !";
        System.out.println(message);
        message = "Bonjour monde...";
        System.out.println(message);
    }
}
```

```
[11:16][Prog pc666 :]$ java ExempleString
Hello World !
Bonjour monde...
```

- contenu **non-modifiable**,
- variable modifiable, simple pointeur.

- **StringBuffer** : afin de stocker une chaîne de caractères modifiable.

```
class ExempleStringBuffer {
    public static void main (String args[]) {
        StringBuffer modifiable = new StringBuffer("Hello World !");
        System.out.println(modifiable);
        modifiable.setCharAt(6, 'M');
        modifiable.setCharAt(8, 'n');
        modifiable.setCharAt(9, 'd');
        modifiable.setCharAt(10, 'e');
        System.out.println(modifiable);
    }
}
```

```
[11:20][Prog pc666 :]$ java ExempleStringBuffer
Hello World !
Hello Monde !
```

- contenu modifiable,
- variable modifiable, simple pointeur.

- 1 Aperçu & Historique
- 2 Types primitifs, opérateurs, conversions, constantes
  - Types primitifs
  - Opérateurs
  - Conversions de types
  - Constantes en JAVA
- 3 Tableaux et chaînes de caractères
  - Tableaux
  - Chaînes de caractères
- 4 Structures de contrôle
- 5 Classes, champs et méthodes
- 6 Concepts de la programmation orientée objet
  - Modélisation
  - Encapsulation des données
- 7 Bonnes pratiques et outils
  - Bonnes pratiques
  - Outils pour la POO et JAVA

# Structures de contrôle conditionnelle

## Avec alternative simple :

```
class StructureIfElse {
    public static void main(String args[]) {
        float moyenneExamen = 13.75f;
        String decision = "";
        if (moyenneExamen < 10.0f) {
            decision = "Ajourné";
        }
        else {
            decision = "Reçu";
        }
        System.out.println(decision);
    }
}
```

```
[14:25] [Prog pc666 :]$ java StructureIfElse
Reçu
```

## Remarques :

- ▶ La partie else est optionnelle.
- ▶ Lorsque le bloc if ou else ne comporte qu'une unique instruction alors les accolades sont inutiles.
- ▶ Pour des tests simples et succincts on pourra utiliser l'opérateur ternaire.

## Avec alternatives multiples :

```
class StructureIfElseIf {
    public static void main(String args[]) {
        float moyenneExamen = 13.75f;
        String mention = "";
        if (moyenneExamen < 10.0f) {
            mention = "Ajourné";
        }
        else if (moyenneExamen < 12.0f) {
            mention = "Admis";
        }
        else if (moyenneExamen < 14.0f) {
            mention = "Admis - assez bien";
        }
        else if (moyenneExamen < 16.0f) {
            mention = "Admis - bien";
        }
        else {
            mention = "Admis - très bien";
        }
        System.out.println(mention);
    }
}
```

```
[15:36] [Prog pc666 :]$ java StructureIfElseIf
Admis - assez bien
```

✓ Plus lisible que des structures if et else imbriquées.

# Structure de contrôle à choix multiples

Un switch permet de « brancher » directement sur un groupe d'instructions selon la valeur d'une étiquette.

```
class StructureSwitch {
    public static void main(String args[]) {
        String corpsCeleste = args[0];
        switch (corpsCeleste) {
            case "Mercure" :
            case "Vénus" :
            case "Terre" :
            case "Mars" :
                System.out.println(corpsCeleste +
                    " est une planète tellurique.");
                break;
            case "Jupiter" :
            case "Saturne" :
            case "Uranus" :
            case "Neptune" :
                System.out.println(corpsCeleste +
                    " est une planète gazeuse.");
                break;
            default :
                System.out.println(corpsCeleste +
                    " n'est pas une planète du système solaire.");
        }
    }
}
```

```
[16:05][Prog pc666 :]$ java StructureSwitch Mars
Mars est une planète tellurique.
[16:05][Prog pc666 :]$ java StructureSwitch Pluton
Pluton n'est pas une planète du système solaire.
```

## Utiliser String args[]

Permet de récupérer les arguments de la ligne de commande.

## Remarques :

- ▶ Penser à mettre des instructions break ainsi qu'une étiquette default.
- ▶ Possibilité d'imbriquer en cascade les étiquettes case (comme ici).
- ▶ Depuis JAVA 7, possibilité d'utiliser des chaînes de caractères comme étiquette des case (comme ici).

# Structure de contrôle répétitive for

Pour répéter un bloc d'instructions, on pourra utiliser une boucle `for`. Il existe deux formes :

## ► La forme classique

```
class StructureFor {  
    public static void main(String args[]) {  
        String tabCorpsCelestes[] = {"Uranus", "Ceres",  
                                       "Pluton", "Io"};  
        for (int i = 0; i < tabCorpsCelestes.length; i++)  
            System.out.println(tabCorpsCeleste[i] +  
                               " est dans le système solaire.");  
    }  
}
```

```
[17:09][Prog pc666 :]$ java StructureFor  
Uranus est dans le système solaire.  
Ceres est dans le système solaire.  
Pluton est dans le système solaire.  
Io est dans le système solaire.
```

## ► La forme « *for each* »

```
class StructureForEach {  
    public static void main(String args[]) {  
        String tabCorpsCelestes[] = {"Uranus", "Ceres",  
                                       "Pluton", "Io"};  
        for (String cs : tabCorpsCelestes) {  
            System.out.println(cs + " est dans" +  
                               " le système solaire.");  
        }  
    }  
}
```

Cette forme est très pratique pour les structures comme les tableaux.

# Structures de contrôle répétitives while et do...while

Pour répéter un bloc d'instructions, on peut également utiliser les structures de contrôle ayant les formes suivantes :

## ► La forme while

```
import java.util.Scanner;

class StructureWhile {
    public static void main(String args[]) {
        Scanner clavier = new Scanner(System.in);
        while (clavier.hasNextInt()) {
            System.out.println(clavier.nextInt());
        }
    }
}
```

- Éventuellement aucun passage dans la boucle.

## ► La forme do...while

```
import java.util.Scanner;

class StructureDoWhile {
    public static void main(String args[]) {
        Scanner clavier = new Scanner(System.in);
        int entierLu;
        do {
            entierLu = clavier.nextInt();
            System.out.println(entierLu);
        } while (entierLu > 0);
    }
}
```

- Au moins un passage dans la boucle.

# Rupture du flot d'instructions : break & continue – 1/2

Possibilité de modifier le flot d'instructions avec l'instruction :

- ▶ **break** pour mettre fin à la structure de contrôle courante la plus interne.

```
...
for (int i = 0; i < tabNote.length; i++) {
    if (tabNote[i] >= 0) {
        sommeNote += tabNote[i];
    }
    else {
        System.out.println("ERREUR : note " + i);
        break;
    }
}
...
```

## Branchement inconditionnel : usages

Gestion classique du comportement des structures de contrôle :

- ▶ switch,
- ▶ for,
- ▶ while et do...while.

- ▶ **continue** pour
  - mettre fin à l'itération courante de la structure de contrôle courante la plus interne;
  - passer à l'itération suivante.

```
...
for (int i = 0; i < tabNote.length; i++) {
    if (tabNote[i] >= 0) {
        sommeNote += tabNote[i];
    }
    else {
        System.out.println("AVERTISSEMENT : note "
                           + i + "négative");
        continue;
    }
}
...
```

# Rupture du flot d'instructions : break & continue – 2/2

- Possibilité d'étiquetter les structures de contrôle et d'effectuer des branchements inconditionnels dessus.

⇒ permet de sortir de plusieurs niveaux d'imbrication.

```
class BreakEtContinue {
    public static void main(String args[]) {
        int tabNotesExamens[][] = { // Tableau bidimensionnel.
            {10, 11, 17, 8, 20}, // CC1
            {12, 10, -666, 17, 18}, // CC2
            {12, 10, 11, 17, 18} // CC3
        };

        boucleExterne :
        for (int i = 0; i < tabNotesExamens[0].length; i++) {
            for (int j = 0; j < tabNotesExamens[1].length; j++) {
                if (tabNotesExamens[i][j] < 0) {
                    System.out.println("\nERREUR : [" + i + "," + j + "] = "
                        + tabNotesExamens[i][j]);
                    break boucleExterne;
                }
                System.out.print(tabNotesExamens[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

```
[21:56] [Prog pc666 :]$ java BreakEtContinue
10 11 17 8 20
12 10
ERREUR : [1,2] = -666
```

Valides pour :

- switch,
- for,
- while et do...while.



- 1 Aperçu & Historique
- 2 Types primitifs, opérateurs, conversions, constantes
  - Types primitifs
  - Opérateurs
  - Conversions de types
  - Constantes en JAVA
- 3 Tableaux et chaînes de caractères
  - Tableaux
  - Chaînes de caractères
- 4 Structures de contrôle
- 5 Classes, champs et méthodes
- 6 Concepts de la programmation orientée objet
  - Modélisation
  - Encapsulation des données
- 7 Bonnes pratiques et outils
  - Bonnes pratiques
  - Outils pour la POO et JAVA

# Notion d'objet et de classe

Concept central de la programmation orienté objet (POO) :

- 1 tout est objet,
- 2 les objets sont définis selon un modèle que l'on appelle une classe.

## Une analogie : classe et dictionnaire

Par analogie, une classe est comme tout nom d'un dictionnaire.

Par exemple, le mot « avion » est ce concept intuitif d'un véhicule :

- ▶ doté d'ailes,
- ▶ doté de réacteurs,
- ▶ permettant de se déplacer dans les airs.

Un objet de la classe « avion » est alors une réalisation physique particulière, tout comme le sont :

- ▶ *Air-Force One*,
- ▶ l'avion pour les vols courts et moyens courriers du président français,
- ▶ l'avion que vous avez pris cet été.

# Le concept de classe

- ▶ Une classe est un modèle de données défini par le concepteur comme étant :
  - ① un ensemble de champs permettant de décrire l'état de chaque objet,
  - ② un ensemble de méthodes permettant à tout objet des interactions.
- ▶ Une classe définit également un type (appelé type référence).
  - Le nom de ce type est celui de la classe.
  - Le nom d'une classe commence par une majuscule (usage).

## Analogie – suite

Par exemple, aussi différents soient les avions précédemment cités, ils disposent :

### ① de champs,

- numéro d'immatriculation,
- capacité du réservoir,
- nombre de sièges,
- fabricant...

### ② de méthodes,

- décoller,
- atterrir,
- purger le réservoir,
- communiquer...

# Les champs

Les champs sont les caractéristiques énoncées par le modèle d'une classe :

- ➊ ils sont en nombre fini,
- ➋ ils sont nommés et typés,
- ➌ ils attribuent à chaque objet instancié (*i.e.* créé) selon cette classe des valeurs particulières.

## Analogie – suite

Par exemple, aussi différents soient les avions précédemment cités, ils disposent des mêmes champs (affectés de valeurs différentes) :

### ➊ *Air Force One*,

- numéro d'immatriculation :  
US-123-AZ-B747-200B
- capacité du réservoir :  
240000L
- nombre de sièges : 100 pers.
- fabricant : Boeing...

### ➋ *Air Force Macron*,

- numéro d'immatriculation :  
FR-555-UV-DFALC-7X
- capacité du réservoir :  
2100L
- nombre de sièges : 15 pers.
- fabricant : Dassault...

# Les méthodes

Les méthodes correspondent à la notion de fonction de la plupart des langages.

Cependant, en POO :

- ▶ il faut créer préalablement des objets pour utiliser les méthodes ;

✓ Ce sont les objets qui activent leurs méthodes.

- ▶ les méthodes sont rattachées à un type de donnée particulier, c'est-à-dire à une classe donnée.

✗ Impossibilité pour les objets de la classe Automobile de décoller...

## Une méthode indispensable : la méthode constructeur

En POO, il est nécessaire d'indiquer à l'ordinateur comment doivent être initialisés les champs. C'est le rôle de la méthode dite constructeur.

# Les méthodes – syntaxe

Les méthodes sont les moyens d'action des objets avec eux-mêmes ou d'autres objets :

- 1 elles sont en nombre fini,
- 2 elles sont nommées et typées (*i.e.* valeur de retour et arguments).

Syntaxe générale :

```
typeRetour nomMethode([type1 arg1, type2 arg2,..., typeN argN]) {  
    instructions;  
    [return [resultat];]  
}
```

Appeler une méthode (avec un objet de sa classe) :

```
nomObjet.nomMethode(arg1, arg2,...,argN);
```

La référence `monObjet` doit alors désigner un objet qui existe en mémoire.

# Les méthodes – aspects techniques

- ▶ En JAVA, les arguments sont transmis par valeur :
  - une variable de type primitif fournie en argument sera copiée ;
  - une variable de type référence fournie en argument sera copiée (seulement la référence, l'objet désigné sera le même).
- ▶ Les arguments d'une méthode sont évalués selon l'ordre d'énonciation.

```
void methodeScientifique(int a, int b) {...}  
...  
public static void main(String args[]) {  
    ...  
    int nb = 10;  
    obj.methodeScientifique(nb++, ++nb);  
}
```

- nb++ est d'abord exécuté, puis ++nb lors de l'appel à methodeScientifique().

≠ langage C !

- ▶ La JVM autorise l'usage d'arguments de type primitif ayant un type hiérarchiquement inférieur au type spécifié par la méthode.

```
void methodeCoue(double moyenne) {...}  
...  
public static void main(String args[]) {  
    ...  
    float noteCC = 14f;  
    obj.methodeCoue(noteCC);  
}
```

- noteCC est promu en int lors de l'appel à methodeCoue().

# Objet : déclaration, création et accès aux champs

- ▶ Pour utiliser un objet, il faut déclarer une référence/variable désignant cet objet.

- Syntaxe générale :

```
TypeDeClasse nomObjet;
```

- Exemple ;

```
Scanner clavier;
```

- ▶ Pour utiliser un objet, il faut le créer (appel d'un constructeur) :

- Syntaxe générale :

```
nomObjet = new TypeDeClasse([arg...]);
```

- Exemple ;

```
clavier = new Scanner(System.in);
```

- ▶ Pour accéder aux valeurs des champs d'un objet donné, on utilise l'opérateur « . ».

- Syntaxe générale :

```
nomObjet.champ;
```

- Exemple :

```
Avion a1 = new Avion("FR-75", 100,  
4, "Cessna");  
System.out.println("Fabricant" + a1.fab)  
// Où fab est champ de la classe Avion.
```



# Définir une classe en JAVA – 1/2

On souhaite définir une classe `Identite` pour représenter des identités.

Après modélisation, la classe sera structurée par :

- |  |   |
|--|---|
| <p>❶ les champs,</p> <ul style="list-style-type: none"><li>• nom,</li><li>• prenom,</li><li>• age,</li><li>• adresse,</li><li>• estValide.</li></ul> | <p>❷ de méthodes,</p> <ul style="list-style-type: none"><li>• <code>afficherIdentite()</code>,</li><li>• <code>modifierAdresse()</code>,</li><li>• <code>donnerValidite()</code>.</li></ul> |
|--|---|

Il faut typer correctement tout ceci.

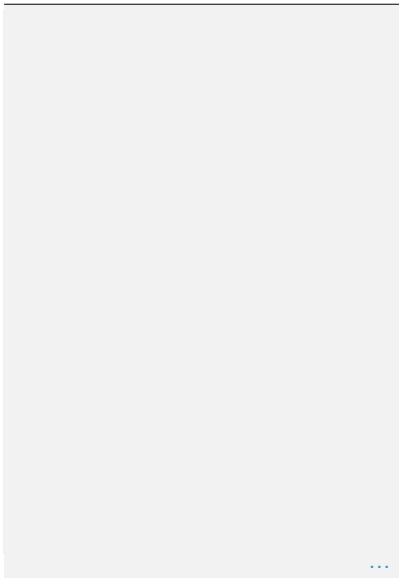
## Remarque

Prendre l'habitude :

- ▶ de nommer les méthodes avec des verbes,
- ▶ d'utiliser des tournures affirmatives (champs et méthodes).

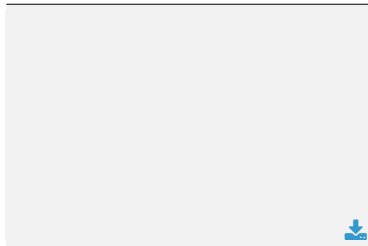
# Définir une classe en JAVA – 2/2

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32



...

33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45



[16:05][Prog pc666 :]\$

# Définir une classe en JAVA – 2/2

```
1 class Identite {
```

2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32

...

```
}
```



33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45

```
[16:05][Prog pc666 :]$
```

# Définir une classe en JAVA – 2/2

```
1  class Identite {  
2      String nom;  
3      String prenom;  
4      int age;  
5      String adresse;  
6      boolean estValide;  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32
```

```
}  
  
}
```

```
[16:05][Prog pc666 :]$
```

# Définir une classe en JAVA – 2/2

```
1  class Identite {  
2      String nom;  
3      String prenom;  
4      int age;  
5      String adresse;  
6      boolean estValide;  
7  
8      Identite(String n, String p, int age,  
9          String adr, boolean v) {  
10         nom = n;  
11         prenom = p;  
12         this.age = age;  
13         adresse = adr;  
14         estValide = v;  
15     }  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32
```


```
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
}
```

[16:05][Prog pc666 :]\$

# Définir une classe en JAVA – 2/2

```
1  class Identite {
2      String nom;
3      String prenom;
4      int age;
5      String adresse;
6      boolean estValide;
7
8      Identite(String n, String p, int age,
9              String adr, boolean v) {
10         nom = n;
11         prenom = p;
12         this.age = age;
13         adresse = adr;
14         estValide = v;
15     }
16
17     void afficherIdentite() {
18         System.out.println(nom + ", " +
19                             prenom + ", " +
20                             age + ", " +
21                             adresse + ", " +
22                             (estValide ? "VAL" : "INV"));
23     }
24
25
26
27
28
29
30
31
32
```

```
33
34
35
36
37
38
39
40
41
42
43
44
45
```



```
[16:05][Prog pc666 :.]$
```

# Définir une classe en JAVA – 2/2

```
1  class Identite {
2      String nom;
3      String prenom;
4      int age;
5      String adresse;
6      boolean estValide;
7
8      Identite(String n, String p, int age,
9              String adr, boolean v) {
10         nom = n;
11         prenom = p;
12         this.age = age;
13         adresse = adr;
14         estValide = v;
15     }
16
17     void afficherIdentite() {
18         System.out.println(nom + ", " +
19                             prenom + ", " +
20                             age + ", " +
21                             adresse + ", " +
22                             (estValide ? "VAL" : "INV"));
23     }
24
25     void modifierAdresse(String nouvelleAdresse) {
26         adresse = nouvelleAdresse;
27     }
28
29     boolean donnerValidite() {
30         return estvalide;
31     }
32 }
```

}



```
[16:05][Prog pc666 :]$
```

# Définir une classe en JAVA – 2/2

```
1 class Identite {
2     String nom;
3     String prenom;
4     int age;
5     String adresse;
6     boolean estValide;
7
8     Identite(String n, String p, int age,
9         String adr, boolean v) {
10         nom = n;
11         prenom = p;
12         this.age = age;
13         adresse = adr;
14         estValide = v;
15     }
16
17     void afficherIdentite() {
18         System.out.println(nom + ", " +
19             prenom + ", " +
20             age + ", " +
21             adresse + ", " +
22             (estValide ? "VAL" : "INV"));
23     }
24
25     void modifierAdresse(String nouvelleAdresse) {
26         adresse = nouvelleAdresse;
27     }
28
29     boolean donnerValidite() {
30         return estvalide;
31     }
32 }
```

```
public static void main(String args[]) {
33
34
35
36
37
38
39
40
41
42
43
44
45
}
```

[16:05][Prog pc666 :]\$





# Définir une classe en JAVA – 2/2

```
1 class Identite {  
2     String nom;  
3     String prenom;  
4     int age;  
5     String adresse;  
6     boolean estValide;  
7  
8     Identite(String n, String p, int age,  
9         String adr, boolean v) {  
10         nom = n;  
11         prenom = p;  
12         this.age = age;  
13         adresse = adr;  
14         estValide = v;  
15     }  
16  
17     void afficherIdentite() {  
18         System.out.println(nom + ", " +  
19             prenom + ", " +  
20             age + ", " +  
21             adresse + ", " +  
22             (estValide ? "VAL" : "INV"));  
23     }  
24  
25     void modifierAdresse(String nouvelleAdresse) {  
26         adresse = nouvelleAdresse;  
27     }  
28  
29     boolean donnerValidite() {  
30         return estvalide;  
31     }  
32 }
```

```
public static void main(String args[]) {  
    Identite id_1, id_2;  
  
  
  
  
}
```

[16:05][Prog pc666 :]\$



# Définir une classe en JAVA – 2/2

```

1  class Identite {
2      String nom;
3      String prenom;
4      int age;
5      String adresse;
6      boolean estValide;
7
8      Identite(String n, String p, int age,
9              String adr, boolean v) {
10         nom = n;
11         prenom = p;
12         this.age = age;
13         adresse = adr;
14         estValide = v;
15     }
16
17     void afficherIdentite() {
18         System.out.println(nom + ", " +
19                             prenom + ", " +
20                             age + ", " +
21                             adresse + ", " +
22                             (estValide ? "VAL" : "INV"));
23     }
24
25     void modifierAdresse(String nouvelleAdresse) {
26         adresse = nouvelleAdresse;
27     }
28
29     boolean donnerValidite() {
30         return estvalide;
31     }
32

```

```

public static void main(String args[]) {
    Identite id_1, id_2;
    id_1 = new Identite("Presley", "Elvis",
                        84, "Memphis", true);
    id_2 = new Identite("Franklin", "Aretha",
                        77, "Detroit", true);
}

```

[16:05][Prog pc666 :]\$



# Définir une classe en JAVA – 2/2

```

1  class Identite {
2      String nom;
3      String prenom;
4      int age;
5      String adresse;
6      boolean estValide;
7
8      Identite(String n, String p, int age,
9              String adr, boolean v) {
10         nom = n;
11         prenom = p;
12         this.age = age;
13         adresse = adr;
14         estValide = v;
15     }
16
17     void afficherIdentite() {
18         System.out.println(nom + ", " +
19                             prenom + ", " +
20                             age + ", " +
21                             adresse + ", " +
22                             (estValide ? "VAL" : "INV"));
23     }
24
25     void modifierAdresse(String nouvelleAdresse) {
26         adresse = nouvelleAdresse;
27     }
28
29     boolean donnerValidite() {
30         return estvalide;
31     }
32

```

```

public static void main(String args[]) {
    Identite id_1, id_2;
    id_1 = new Identite("Presley", "Elvis",
                        84, "Memphis", true);
    id_2 = new Identite("Franklin", "Aretha",
                        77, "Detroit", true);
    id_1.afficherIdentite();
    id_2.afficherIdentite();
    id_1.modifierAdresse("Melun");
    id_1.afficherIdentite();
}

```

[16:05][Prog pc666 :]\$



# Définir une classe en JAVA – 2/2

```

1  class Identite {
2      String nom;
3      String prenom;
4      int age;
5      String adresse;
6      boolean estValide;
7
8      Identite(String n, String p, int age,
9              String adr, boolean v) {
10         nom = n;
11         prenom = p;
12         this.age = age;
13         adresse = adr;
14         estValide = v;
15     }
16
17     void afficherIdentite() {
18         System.out.println(nom + ", " +
19                             prenom + ", " +
20                             age + ", " +
21                             adresse + ", " +
22                             (estValide ? "VAL" : "INV"));
23     }
24
25     void modifierAdresse(String nouvelleAdresse) {
26         adresse = nouvelleAdresse;
27     }
28
29     boolean donnerValidite() {
30         return estvalide;
31     }
32

```

```

33     public static void main(String args[]) {
34         Identite id_1, id_2;
35         id_1 = new Identite("Presley", "Elvis",
36                             84, "Memphis", true);
37         id_2 = new Identite("Franklin", "Aretha",
38                             77, "Detroit", true);
39         id_1.afficherIdentite();
40         id_2.afficherIdentite();
41         id_1.modifierAdresse("Melun");
42         id_1.afficherIdentite();
43         System.out.println(id_1.donnerValidite());
44     }
45

```



[16:05][Prog pc666 :]\$

# Définir une classe en JAVA – 2/2

```

1  class Identite {
2      String nom;
3      String prenom;
4      int age;
5      String adresse;
6      boolean estValide;
7
8      Identite(String n, String p, int age,
9              String adr, boolean v) {
10         nom = n;
11         prenom = p;
12         this.age = age;
13         adresse = adr;
14         estValide = v;
15     }
16
17     void afficherIdentite() {
18         System.out.println(nom + ", " +
19                             prenom + ", " +
20                             age + ", " +
21                             adresse + ", " +
22                             (estValide ? "VAL" : "INV"));
23     }
24
25     void modifierAdresse(String nouvelleAdresse) {
26         adresse = nouvelleAdresse;
27     }
28
29     boolean donnerValidite() {
30         return estvalide;
31     }
32

```

```

33     public static void main(String args[]) {
34         Identite id_1, id_2;
35         id_1 = new Identite("Presley", "Elvis",
36                             84, "Memphis", true);
37         id_2 = new Identite("Franklin", "Aretha",
38                             77, "Detroit", true);
39         id_1.afficherIdentite();
40         id_2.afficherIdentite();
41         id_1.modifierAdresse("Melun");
42         id_1.afficherIdentite();
43         System.out.println(id_1.donnerValidite());
44     }
45

```

[16:05][Prog pc666 :]\$

## Remarques :

- le mot clé this désigne l'objet en cours de construction.
- création des objets.
- appels de méthodes.

# Définir une classe en JAVA – 2/2

```

1  class Identite {
2      String nom;
3      String prenom;
4      int age;
5      String adresse;
6      boolean estValide;
7
8      Identite(String n, String p, int age,
9              String adr, boolean v) {
10         nom = n;
11         prenom = p;
12         this.age = age;
13         adresse = adr;
14         estValide = v;
15     }
16
17     void afficherIdentite() {
18         System.out.println(nom + ", " +
19                             prenom + ", " +
20                             age + ", " +
21                             adresse + ", " +
22                             (estValide ? "VAL" : "INV"));
23     }
24
25     void modifierAdresse(String nouvelleAdresse) {
26         adresse = nouvelleAdresse;
27     }
28
29     boolean donnerValidite() {
30         return estvalide;
31     }
32

```

```

33     public static void main(String args[]) {
34         Identite id_1, id_2;
35         id_1 = new Identite("Presley", "Elvis",
36                             84, "Memphis", true);
37         id_2 = new Identite("Franklin", "Aretha",
38                             77, "Detroit", true);
39         id_1.afficherIdentite();
40         id_2.afficherIdentite();
41         id_1.modifierAdresse("Melun");
42         id_1.afficherIdentite();
43         System.out.println(id_1.donnerValidite());
44     }
45

```

```
[16:05][Prog pc666 :]$ java Identite
```

## Remarques :

- le mot clé this désigne l'objet en cours de construction.
- création des objets.
- appels de méthodes.

# Définir une classe en JAVA – 2/2

```

1  class Identite {
2      String nom;
3      String prenom;
4      int age;
5      String adresse;
6      boolean estValide;
7
8      Identite(String n, String p, int age,
9              String adr, boolean v) {
10         nom = n;
11         prenom = p;
12         this.age = age;
13         adresse = adr;
14         estValide = v;
15     }
16
17     void afficherIdentite() {
18         System.out.println(nom + ", " +
19                             prenom + ", " +
20                             age + ", " +
21                             adresse + ", " +
22                             (estValide ? "VAL" : "INV"));
23     }
24
25     void modifierAdresse(String nouvelleAdresse) {
26         adresse = nouvelleAdresse;
27     }
28
29     boolean donnerValidite() {
30         return estvalide;
31     }
32

```

```

33     public static void main(String args[]) {
34         Identite id_1, id_2;
35         id_1 = new Identite("Presley", "Elvis",
36                             84, "Memphis", true);
37         id_2 = new Identite("Franklin", "Aretha",
38                             77, "Detroit", true);
39         id_1.afficherIdentite();
40         id_2.afficherIdentite();
41         id_1.modifierAdresse("Melun");
42         id_1.afficherIdentite();
43         System.out.println(id_1.donnerValidite());
44     }
45

```

```

[16:05][Prog pc666 :]$ java Identite
Presley, Elvis, 84, Memphis, VAL
Franklin, Aretha, 77, Detroit, VAL
Presley, Elvis, 84, Melun, VAL
true

```

## Remarques :

- le mot clé this désigne l'objet en cours de construction.
- création des objets.
- appels de méthodes.

# La méthode constructeur

Il s'agit d'une méthode construisant un objet en mémoire selon d'éventuels paramètres et retournant l'adresse de cet objet.

► Appel au constructeur :

```
TypeDeClasse nomObjet = new TypeDeClasse([arg...]);
```

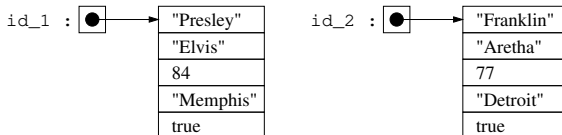
► Définition du constructeur :

- inutile de préciser le type de retour,
- inutile de mettre un return.

► Exemple :

```
35 id_1 = new Identite("Elvis", "Presley",  
36                      84, "Memphis", true);  
37 id_2 = new Identite("Franklin", "Aretha",  
38                      77, "Detroit", true);
```

Après appel au constructeur, deux objets ont été alloués dynamiquement en mémoire par la JVM :



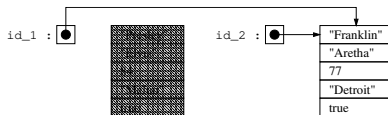


# Durée de vie des objets

## ► Supposons que :

```
...
public static void main(String args[]) {
    Identite id_1, id_2;
    id_1 = new Identite("Presley", "Elvis",
                        84, "Memphis", true);
    id_2 = new Identite("Franklin", "Aretha",
                        77, "Detroit", true);
    ...
    System.out.println(id_1.estValide());
    id_1 = id_2;
    ...
}
```

## ► Après l'instruction surlignée, nous aurions en mémoire :



L'objet initialement référencé par `id_1` ne l'est plus.

Il sera **automatiquement détruit** par le « ramasse-miettes » de la JVM.

## Gestion de la mémoire en JAVA

La gestion de la mémoire est simplifiée :

- par l'opérateur `new` (pas d'allocation manuelle),
- par le ramasse-miettes (plus de gestion du programmeur).

Le code est beaucoup plus sûr.

# Champs et méthodes d'instance

Un objet créé selon le modèle d'une classe est aussi appelé instance de cette classe. Chaque instance :

- ▶ a son propre espace mémoire (et ses propres valeurs de champs),
- ▶ peut appeler les méthodes de sa classe.

Tout objet/instance dispose à l'exécution de :

## ① champs d'instance

- propres à l'instance,
- exemple :

```
id_1.age = 42;  
id_1.adresse = id_2.adresse;
```

## ② méthodes d'instance

- utilisables par une instance,
- exemple :

```
id_2.afficherIdentite();  
id_2.modifierAdresse("Cajarc")
```

Dans les méthodes d'instance de la classe courante, on peut directement modifier ces champs ou appeler ces méthodes.

```
Identite(String n, String p, int age,  
        String adr, boolean v) {  
    nom = n;  
    prenom = p;  
    this.age = age;  
    ...  
    afficherIdentite(); // Inutile d'écrire this.afficherIdentite().  
}
```

# Champs de classe – le mot clé static

- *Quid* de la notion de variable « globale » en JAVA ?

⇒ définir un champ static.

- Exemple :

```
class Identite {
    static int cptId = 0;
    int numId;
    String nom;
    ...

    Identite(String n, String p, int age,
              String adr, boolean v) {
        numId = cptId++;
        nom = n;
        ...
    }

    void afficherIdentite() {
        System.out.println(numId + ", " +
                           nom + ", " +
                           ...
    }
    ...
}
```

- Syntaxe générale :

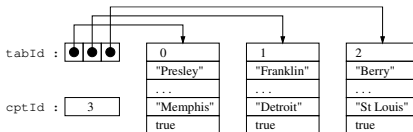
```
static TypePrimitif champ = valeur;
```

```
...
public static void main(String args[]) {
    Identite tabId[] = {
        new Identite("Presley", "Elvis",
                     84, "Memphis", true),
        new Identite("Franklin", "Aretha",
                     77, "Detroit", true),
        new Identite("Berry", "Chuck",
                     90, "St Louis", true),
    };
    for (Identite id : tabId) {
        id.afficherIdentite();
    }
}
}
```

```
[11:14][Prog pc666 :]$ java Identite
0, Presley, Elvis, 84, Memphis, VAL
1, Franklin, Aretha, 77, Detroit, VAL
2, Berry, Chuck, 90, St Louis, VAL
```

# Champs de classe – le mot clé static

- ▶ En mémoire, un champ statique existe en un seul exemplaire.



- ▶ Bloc d'initialisation statique.

```
import java.util.Scanner;
class BlocStatique {
    static int tab[];
    static { // Taille définie à l'exécution.
        Scanner clavier = new Scanner(System.in);
        System.out.println("Combien d'éléments ?");
        int nbElem = clavier.nextInt();
        tab = new int[nbElem];
        for (int i = 0; i < nbElem; i++) {
            tab[i] = clavier.nextInt();
        }
    }

    public static void main(String args[]) {
        System.out.println("Contenu de tab[] :");
        for (int i = 0; i < tab.length; i++) {
            System.out.print(tab[i] + " ");
        }
        System.out.println();
    }
}
```

```
[15:18] [Prog pc666 :]$ java BlocStatique
Combien d'éléments ?
3
10 15 20
Contenu de tab[] :
10 15 20
```

## Initialisation d'un champ statique

- ▶ Implicitement à zéro.
- ▶ Explicitement selon une valeur (au plus tard par le constructeur).
- ▶ Explicitement à sa déclaration si ce champ est final.

# Méthodes de classe – le mot clé static

- ▶ Comment faire pour calculer une simple somme de carrés ?  
Doit-on créer une classe, instancier des objets, puis appeler ces méthodes ?

✗ Peu pratique et lourd !

✓ JAVA prévoit des  
« méthodes-fonctions »

- ▶ Syntaxe générale :

- Définition :

```
static [TypePrimitif] nomMethode(arg...) {
    instructions;
    [return [resultat];]
}
```

- ▶ Exemple :

```
import java.util.Scanner;

class ExempleMethStatique {
    static int sommerCarres(int n) {
        int somme = 0;
        for (int i = 1; i <= n; i++) {
            somme += i * i;
        }
        return somme;
    }
}
```

...

- Usage : **ne manipule que des champs statiques.**

```
nomMethode(arg...); // dans la classe courante A.
A.nomMethode(arg...); // dans une autre classe.
```

```
public static void main(String args[]) {
    Scanner clavier = new Scanner(System.in);
    int nbLu = clavier.nextInt();
    System.out.println("Somme des carrés de 1 à " + nbLu
        + " : " + sommerCarres(nbLu));
}
}
```



```
[16:08][Prog pc666 :]$ java ExempleMethStatique
4
Somme des carrés de 1 à 4 : 30
```

# Champs et méthodes statiques issus d'une autre classe

- ▶ La classe `Math`<sup>1</sup> de l'API JAVA comporte :
  - ① les constantes mathématiques  $\pi$  et  $e$ ,
  - ② une cinquantaine de méthodes mathématiques et de manipulation de nombres flottants.

Tous ces champs et méthodes sont déclarés en `static`.

- ▶ Utiliser les champs ou méthodes statiques d'une autre classe nécessite de les préfixer par le nom de la classe.
- ▶ Exemple avec la classe `Math` :

```
class ExempleMathematique {  
    public static void main(String args[]) {  
        double valeurs[] = {Math.PI / 6, Math.PI / 4, Math.PI / 3};  
        for (double val : valeurs) {  
            System.out.printf("%.3f\n", Math.cos(val));  
        }  
    }  
}
```

```
[12:37] [Prog pc666 :]$ java ExempleMathematique  
0.866  
0.707  
0.500
```

Calcul de  $\cos(\pi/6)$ ,  $\cos(\pi/4)$  et  $\cos(\pi/3)$ .

<sup>1</sup><https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html>.

# Bloc d'instructions & variables locales

C'est un ensemble d'instructions délimité par des accolades.

```
class BlocInstructions {
    public static void main (String args[]) {
        int tabNote[] = {6, 2, 13, 15, 10, 9, 0, 0, 9, 19};
        int nbElevésRecalés = 0;
        int nbElevésAvecMention = 0;
        for (int i = 0; i < tabNote.length; i++) {
            if (tabNote[i] < 10) {
                nbElevésRecalés++;
                System.out.println("Eleve " + i + " recalé.");
            }
            else {
                System.out.println("Eleve " + i + " admis.");
                if (tabNote[i] >= 12) {
                    nbElevésAvecMention++;
                }
            }
        }
        System.out.println("Taux d'échec : " +
            100.0f * nbElevésRecalés / tabNote.length);
        System.out.println("Taux avec mention : " +
            100.0f * nbElevésAvecMention / tabNote.length);
    }
}
```

## Bloc : portée & durée de vie des variables locales

- ▶ Portée : celle du bloc.
- ▶ Durée de vie : celle du bloc (variable stockée en mémoire sur une pile).

# Bloc : le problème du « shadowing »

Une erreur courante et parfois pénible à déterminer consiste à redéfinir par mégarde un champ dans un bloc.

```
1 class Identite {  
2     String nom;  
3     String prenom;  
4     int age;  
5     String adresse;  
6     boolean estValide;  
7  
8     Identite(String n, String p, int age,  
9         String adr, boolean v) {  
10        String nom = n; // au lieu de nom = n;  
11        ...  
12    }  
13    ...  
14 }
```

Malgré une compilation sans problème, ce programme rencontrera un problème :

- ▶ Le constructeur n'affectera jamais `n` au champ `nom`.
- ▶ La variable de la ligne 10 est locale au constructeur et vient faire « ombrage » au champ du même nom.
- ▶ Elle n'existe que provisoirement sur la pile.

⇒ bien ausculter son code

Généralisation : faire référence à un champ ombragé – astuce/rappel

`this.champ` désignera toujours un champ de l'objet, même ombragé.



# Récapitulatif des types de variable : portée & durée de vie

Voici un récapitulatif des catégories de variables en JAVA :

<u>Catégorie</u>	<u>Portée</u>	<u>Durée de vie</u>
Champ de classe	La classe courante	Celle du programme
Champ d'instance	La classe courante	Celle de l'objet
Variable de bloc	Le bloc courant	Celle du bloc

La compréhension des règles de portée et de durée de vie est essentielle !

- 1 Aperçu & Historique
- 2 Types primitifs, operateurs, conversions, constantes
  - Types primitifs
  - Opérateurs
  - Conversions de types
  - Constantes en JAVA
- 3 Tableaux et chaînes de caractères
  - Tableaux
  - Chaînes de caractères
- 4 Structures de contrôle
- 5 Classes, champs et méthodes
- 6 Concepts de la programmation orientée objet
  - Modélisation
  - Encapsulation des données
- 7 Bonnes pratiques et outils
  - Bonnes pratiques
  - Outils pour la POO et JAVA

# La programmation orientée objet (POO) – penser objet

En POO, une grande partie de la difficulté consiste en la modélisation.

- ▶ Il faut déterminer et caractériser les différentes classes, ainsi que leurs relations.
- ▶ Penser objet, c'est se mettre à la place de chaque objet et identifier :
  - ❶ ce qui le caractérise (*i.e.* ses champs) ;
  - ❷ les interactions qu'il aura avec son environnement (*i.e.* ses méthodes).
- ▶ Respecter au mieux le principe d'encapsulation (voir plus loin).

La modélisation est une phase de conception bien distincte de celle de la programmation. Elle est même indépendante du langage JAVA !

Mauvaise modélisation = Mauvais programme

# La modélisation en POO - les diagrammes UML

Utilisation de diagrammes UML (*Unified Modeling Language*).

Exemple de la modélisation d'un ordinateur :

Quelles sont les caractéristiques d'un ordinateur ?

- 1 Processeur
- 2 Fréquence du bus
- 3 Prix

Quelles sont les actions d'un ordinateur ?

- 1 calculer()
- 2 diffuser()

Ordinateur
processeur
freqBus
prix
calculer()
diffuser()

FIGURE – Diagramme UML correspondant.

# La modélisation en POO – relation de composition

Une classe est dite en composition avec d'autres classes si parmi ses champs figurent des objets d'autres classes. De plus, un objet de cette classe :

- ▶ désigne un ensemble d'objets dont chacun est une partie de l'objet ;
- ▶ n'existe que si toutes ses parties existent.

Relation forte entre chacun des composants de l'objet.

Raffinons le modèle avec une classe Processeur.

- 1 Marque
- 2 Fréquence du CPU
- 3 Nombre de caches

Quelles sont les actions d'un processeur ?

- 1 additionner()
- 2 multiplier()

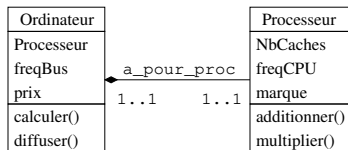


FIGURE – Diagramme UML correspondant.

# La modélisation en POO – relation d'agrégation

Une classe est dite en agrégation avec une autre classe si parmi ses champs figurent des objets d'autres classes non indispensables à l'objet.

- ▶ Relation plus faible entre les objets que celle d'une relation de composition.
- ▶ Durées de vie des objets indépendantes les unes des autres.

Raffinons le modèle avec une classe Imprimante.

- 1 Prix
- 2 Marque
- 3 Encre

Quelles sont les actions d'une imprimante ?

- 1 imprimer()
- 2 agraffer()

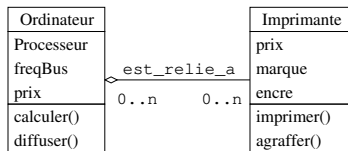


FIGURE – Diagramme UML correspondant.

# La modélisation en POO - relation d'association

Une classe est dite en association avec d'autres classes si certaines de ces méthodes utilisent des objets d'autres classes.

- ▶ il n'y a donc aucune relation au niveau des champs.

Relation faible et uniquement au niveau des interactions.

Raffinons le modèle avec une classe Personne.

- 1 Nom
- 2 Prénom
- 3 Âge

Quelles sont les relations d'une personne avec un ordinateur :

- 1 acheter ordinateur
- 2 revendre ordinateur

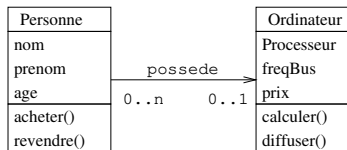
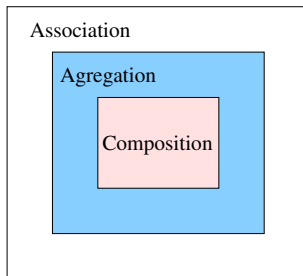


FIGURE – Diagramme UML correspondant.

# La modélisation en POO - synthèse des relations

Schématiquement, la force des relations se représente ainsi :



- ▶ Association : lien faible.
- ▶ Agrégation : lien existant et cependant durée de vie indépendante.
- ▶ Composition : lien étroit et durée de vie dépendante.



# Le principe d'encapsulation – 1/2

En POO pure :

- 1 toutes les données d'un objet lui sont propres,
- 2 seul l'objet est en mesure de les modifier.

En JAVA, il existe une certaine souplesse grâce aux modificateurs d'accès.

<code>public</code>	accessible depuis toute classe.
<code>sans modificateur</code> (visibilité paquetage)	accessible depuis toutes les classes du paquetage ( <i>cf.</i> cours suivant).
<code>protected</code>	idem paquetage & accessible depuis les classes « héritantes » ( <i>cf.</i> cours suivant).
<code>private</code>	accessible seulement depuis la classe hôte.

**Tableau** – Modificateurs d'accessibilité pour un champ ou une méthode.


# Le principe d'encapsulation – 2/2

Pour permettre l'accès à un champ, on met en place des méthodes dites :


► getter : accéder à un champ depuis une autre classe.

► setter : modifier la valeur d'un champ depuis une autre classe.

```
class IdSecrete {  
    private String nom;  
    private int age;  
  
    public IdSecrete(String n, int a) {...}  
  
    public String getNom() { // getter du champ nom.  
        return nom;  
    }  
  
    public int getAge() { // getter du champ age.  
        return age;  
    }  
}  
  
class ExempleGetter {  
    public static void main(String args[]) {  
        IdSecrete id = new IdSecrete("Coltrane", 40);  
        System.out.println(id.getNom());  
        System.out.println(id.getAge());  
    }  
}
```



```
class IdSecrete {  
    private String nom;  
    private int age;  
  
    public IdSecrete(String n, int a) {...}  
  
    public void setNom(String n) { // setter du champ nom.  
        nom = n;  
    }  
  
    public void setAge(int a) { // setter du champ age.  
        age = a;  
    }  
}  
  
class ExempleSetter {  
    public static void main(String args[]) {  
        IdSecrete id = new IdSecrete("Coltrane", 40);  
        id.setNom("Mingus"); // Modif. du champ nom de id.  
        id.setAge(56); // Modif. du champ age de id.  
    }  
}
```



getter/setter ne sont généralement pas définies en private.

- 1 Aperçu & Historique
- 2 Types primitifs, opérateurs, conversions, constantes
  - Types primitifs
  - Opérateurs
  - Conversions de types
  - Constantes en JAVA
- 3 Tableaux et chaînes de caractères
  - Tableaux
  - Chaînes de caractères
- 4 Structures de contrôle
- 5 Classes, champs et méthodes
- 6 Concepts de la programmation orientée objet
  - Modélisation
  - Encapsulation des données
- 7 Bonnes pratiques et outils
  - Bonnes pratiques
  - Outils pour la POO et JAVA

# Les identifiants

« *Mal nommer les choses, c'est ajouter au malheur du monde.* »  
[Albert Camus].

Conseils :

- ▶ utiliser des noms explicites (un bon editeur/IDE fera la complétion automatique);
- ▶ utiliser la notation dite **CamelCase** ou **lowerCamelCase**;
- ▶ utiliser des noms courts pour des variables de faible portée (exemple `i, j...` pour les indices des boucles `for`).

Un code bien écrit se comprend en faisant une lecture à haute voix.

À cela s'ajoutent les contraintes du langage lui-même :

- ▶ en un seul mot;
- ▶ commence par une lettre ou par le caractère '`_`', mais pas un nombre;
- ▶ sensible à la casse (*i.e.* maj/min);
- ▶ ne pas utiliser les mots réservés du langage.

# Écrire du JAVA avec le style

La présentation visuelle du code permet d'améliorer sa lisibilité, sa prise en main et sa maintenance.

Quelques bonnes pratiques :

- ▶ Bien nommer les champs, méthodes et variables.
- ▶ Bien espacer les opérateurs binaires et utiliser des parenthèses.
- ▶ Bien indenter son code.
- ▶ Bien documenter son code.

Le langage JAVA dispose de règles et conseils de présentation :

<https://www.oracle.com/technetwork/java/codeconventions-150003.pdf>

Certaines entreprises définissent leur propre convention :

<https://google.github.io/styleguide/javaguide.html>

# Structure de contrôle : que choisir ?

Les programmes sont faits pour être lus par d'autres programmeurs, et donc entretenus. Comment choisir une...

## ▶ structure conditionnelle ?

- Si les valeurs prises sont finies et connues du programmeurs, préférer un `switch`;
- Sinon préférer les structures `if else`.
- Si l'alternative est uniligne, s'autoriser l'opérateur ternaire `'?'`.

## ▶ structure répétitive ?

- Si le nombre d'itérations est connu, préférer un `for`;
- Sinon préférer les structures `while` et `do...while`.

Il faut écrire de façon lisible, fluide et intuitive !

# Blocs imbriqués

- ▶ Éviter d'imbriquer trop de structures de contrôle.

✗ Code complexe, difficilement déboguable et surtout inefficent.

✓ Se limiter à 80 colonnes de largeur (cf. réglage éditeur).

- ▶ Ne pas abuser des branchements inconditionnels `break` et `continue`.

✗ Programmation « spaghetti » ...

- ▶ En JAVA, pas d'instruction `goto`... car **goto heee111 !**



FIGURE – Auteur et source <https://xkcd.com/292/>

# Programmer, c'est aussi avoir les bons réflexes

- ▶ Tout savoir n'est pas très utile.

✓ la programmation est aussi une affaire de réflexes.

Un programmeur aura le réflexe :

- 1 de bien étudier le problème donné,
  - 2 de chercher dans la documentation (puis sur la toile),
  - 3 de faire des tests pour bien comprendre.
- ▶ Quand on apprend à programmer, ne pas hésiter à expérimenter pour répondre soi-même à ses propres questions.

Exemple : quelle est la valeur attribuée à une variable non initialisée ?

```
class TestVarInit {  
    public static void main(String args[]) {  
        int i;  
        System.out.println("i vaut : " + i);  
    }  
}
```

```
[15:05][Prog pc666 :]$ javac TestVarInit.java  
TestVarInit.java:4: error: variable i might not have been initialized  
    System.out.println("i vaut : " + i);  
                        ^  
1 error
```

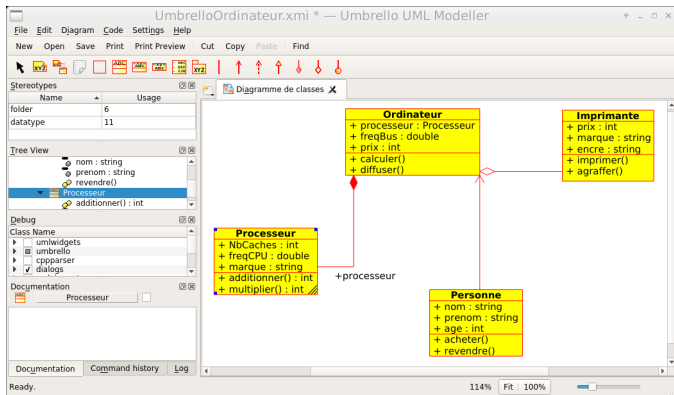
Ce n'est pas une raison pour ignorer presque tout !



# Diagrammes UML : l'outil UMBRELLO

UMBRELLO est un logiciel libre permettant :

- 1 d'éditer des diagrammes UML complets,
- 2 de générer les trames des classes conçues (*i.e.* des fichiers .java).



# L'API JAVA

L'API JAVA est particulièrement riche !

<https://docs.oracle.com/javase/8/docs/api/>

Plusieurs milliers de classes proposées :

- ▶ pour faire des calculs (Math),
- ▶ pour manipuler des chaînes de caractères (String et StringBuffer),
- ▶ pour les structures de données élémentaires ou évoluées,
- ▶ pour le réseau,
- ▶ pour les *threads*,
- ▶ pour les interfaces graphiques (SWING vs. FX)...

Il est impossible/inutile de tout connaître !

# La documentation – l'exemple de l'API JAVA

L'API JAVA est particulièrement riche... et bien documentée !

Java™ Platform Standard Ed. 8

OVERVIEW

PACKAGE

CLASS

USE

TREE

DEPRECATED

INDEX

HELP

Java™ Platform Standard Ed. 8

All Classes

All Profiles

Packages

java.applet

java.awt

java.awt.color

java.awt.datatransfer

java.awt.dnd

java.awt.event

java.awt.font

All Classes

AbstractAction

AbstractAnnotationValueVisitor6

AbstractAnnotationValueVisitor7

AbstractAnnotationValueVisitor8

AbstractBorder

AbstractButton

AbstractCellEditor

AbstractChronology

AbstractCollection

AbstractColorChooserPanel

AbstractDocument

AbstractDocument.AttributeContext

AbstractDocument.Content

AbstractDocument.ElementEdit

AbstractElementVisitor6

AbstractElementVisitor7

AbstractElementVisitor8

AbstractExecutorService

AbstractInterruptibleChannel

AbstractLayoutCache

AbstractLayoutCache.NodeDimensions

AbstractList

AbstractListModel

AbstractMap

AbstractMap.SimpleEntry

PREV CLASS

NEXT CLASS

FRAMES

NO FRAMES

ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD

DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3

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

Constructors

Constructor and Description

Object()

Method Summary

All Methods

Instance Methods

Concrete Methods

Modifier and Type

Method and Description

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.

Contient :

- Description
- Champs
- Constructeurs
- Méthodes
- Exemples

Astuce :

- *frames* (permet de tout voir),
- *no frames* (permet de se concentrer sur une classe).

# La documentation – l'exemple de l'API JAVA

L'API JAVA est particulièrement riche... et bien documentée !

OVERVIEW PACKAGE **CLASS** USE TREE DEPRECATED INDEX HELP Java™ Platform Standard Ed. 8

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD | DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3  
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

**Constructors**

Constructor and Description
Object()

### Method Summary

All Methods	Instance Methods	Concrete Methods
Modifier and Type	Method and Description	
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.

Contient :

- ▶ Description
- ▶ Champs
- ▶ Constructeurs
- ▶ Méthodes
- ▶ Exemples

Astuce :

- ▶ *frames* (permet de tout voir),
- ▶ *no frames* (permet de se concentrer sur une classe).

# La documentation – l'outil JAVADOC

JAVADOC<sup>1</sup>, issu du *Java Development Kit (JDK)*, génère une documentation au format `html` conforme à celle de l'API à l'aide de balises<sup>2</sup> :

- 1 des balises de commentaires spécifiques obligatoires,

```
/**
 * Commentaire JAVADOC.
 * Notez que :
 * a) la première balise comporte une double étoile,
 * b) chaque ligne commence par une étoile.
 */
```

- 2 des balises de mise en forme `html` ordinaires,

```
<p> Commentaire assez long et élaboré. </p>
```

- 3 des balises JAVADOC parmi :

- balises classiques – `@param`, `@return`, `@see...` toutes positionnées en fin d'un commentaire JAVADOC, à raison d'une par ligne,
- balises dites *inlinées* – `{@nomBalise}` pour les autres.

---

<sup>1</sup> Documentation – <https://docs.oracle.com/javase/10/javadoc/toc.htm>

<sup>2</sup> Liste exhaustive – <https://docs.oracle.com/javase/8/docs/technotes/tools/unix/javadoc.html#CHDBEFIF>

# La documentation – exemple avec JAVADOC

```
1  /**
2   * @author O.Marchetti (mon e-mail)
3   * @version 1.0
4   */
5  public class HelloWorld {
6      /**
7       * Cette méthode est le point d'entrée du
8       * programme.
9       *
10      * <p> Ici, on mettra un commentaire plus
11      * long si nécessaire. </p>
12      *
13      * @param args est un tableau de type
14      * String désignant les chaînes de
15      * caractères présentes sur la ligne de
16      * commande.
17      *
18      * @return un entier indiquant que tout
19      * s'est bien passé.
20      */
21      public static int main(String args[]) {
22          System.out.println("Hello world !");
23          return 0;
24      }
25  }
```

```
[10:19][Prog pc666 :]$
```

Commenter :

- ▶ chaque classe,
- ▶ chaque champ,
- ▶ chaque méthode.

## Pour bien restituer les caractères accentués (GNU-LINUX)

Au sein du fichier `.bashrc` (à la racine de votre compte), on positionnera correctement la variable du shell BASH nommée `JAVA_TOOL_OPTIONS` (cf. première ligne du terminal ci-dessus).

# La documentation – exemple avec JAVADOC

```
1  /**
2   * @author O.Marchetti (mon e-mail)
3   * @version 1.0
4   */
5  public class HelloWorld {
6      /**
7       * Cette méthode est le point d'entrée du
8       * programme.
9       *
10      * <p> Ici, on mettra un commentaire plus
11      * long si nécessaire. </p>
12      *
13      * @param args est un tableau de type
14      * String désignant les chaînes de
15      * caractères présentes sur la ligne de
16      * commande.
17      *
18      * @return un entier indiquant que tout
19      * s'est bien passé.
20      */
21      public static int main(String args[]) {
22          System.out.println("Hello world !");
23          return 0;
24      }
25  }
```

```
[10:19][Prog pc666 :]$ grep JAVA ~/.bashrc
```

Commenter :

- ▶ chaque classe,
- ▶ chaque champ,
- ▶ chaque méthode.

## Pour bien restituer les caractères accentués (GNU-LINUX)

Au sein du fichier `.bashrc` (à la racine de votre compte), on positionnera correctement la variable du shell BASH nommée `JAVA_TOOL_OPTIONS` (cf. première ligne du terminal ci-dessus).

# La documentation – exemple avec JAVADOC

```
1  /**
2   * @author O.Marchetti (mon e-mail)
3   * @version 1.0
4   */
5  public class HelloWorld {
6      /**
7       * Cette méthode est le point d'entrée du
8       * programme.
9       *
10      * <p> Ici, on mettra un commentaire plus
11      * long si nécessaire. </p>
12      *
13      * @param args est un tableau de type
14      * String désignant les chaînes de
15      * caractères présentes sur la ligne de
16      * commande.
17      *
18      * @return un entier indiquant que tout
19      * s'est bien passé.
20      */
21      public static int main(String args[]) {
22          System.out.println("Hello world !");
23          return 0;
24      }
25  }
```

```
[10:19][Prog pc666 :]$ grep JAVA ~/.bashrc
export JAVA_TOOL_OPTIONS=-Dfile.encoding=UTF-8
[10:19][Prog pc666 :]$
```

Commenter :

- ▶ chaque classe,
- ▶ chaque champ,
- ▶ chaque méthode.

## Pour bien restituer les caractères accentués (GNU-LINUX)

Au sein du fichier `.bashrc` (à la racine de votre compte), on positionnera correctement la variable du shell BASH nommée `JAVA_TOOL_OPTIONS` (cf. première ligne du terminal ci-dessus).



# La documentation – exemple avec JAVADOC

```

1  /**
2   * @author O.Marchetti (mon e-mail)
3   * @version 1.0
4   */
5  public class HelloWorld {
6      /**
7       * Cette méthode est le point d'entrée du
8       * programme.
9       *
10      * <p> Ici, on mettra un commentaire plus
11      * long si nécessaire. </p>
12      *
13      * @param args est un tableau de type
14      * String désignant les chaînes de
15      * caractères présentes sur la ligne de
16      * commande.
17      *
18      * @return un entier indiquant que tout
19      * s'est bien passé.
20      */
21      public static int main(String args[]) {
22          System.out.println("Hello world !");
23          return 0;
24      }
25  }

```

```

[10:19][Prog pc666 :]$ grep JAVA ~/.bashrc
export JAVA_TOOL_OPTIONS=-Dfile.encoding=UTF-8
[10:19][Prog pc666 :]$ javadoc -d MaDoc HelloWorld.java

```

Commenter :

- ▶ chaque classe,
- ▶ chaque champ,
- ▶ chaque méthode.

**Pour bien restituer les caractères accentués (GNU-LINUX)**

Au sein du fichier .bashrc (à la racine de votre compte), on positionnera correctement la variable du shell BASH nommée `JAVA_TOOL_OPTIONS` (cf. première ligne du terminal ci-dessus).

# La documentation – exemple avec JAVADOC

```

1  /**
2   * @author O.Marchetti (mon e-mail)
3   * @version 1.0
4   */
5  public class HelloWorld {
6      /**
7       * Cette méthode est le point d'entrée du
8       * programme.
9       *
10      * <p> Ici, on mettra un commentaire plus
11      * long si nécessaire. </p>
12      *
13      * @param args est un tableau de type
14      * String désignant les chaînes de
15      * caractères présentes sur la ligne de
16      * commande.
17      *
18      * @return un entier indiquant que tout
19      * s'est bien passé.
20      */
21      public static int main(String args[]) {
22          System.out.println("Hello world !");
23          return 0;
24      }
25  }

```

```

[10:19][Prog pc666 :]$ grep JAVA ~/.bashrc
export JAVA_TOOL_OPTIONS=-Dfile.encoding=UTF-8
[10:19][Prog pc666 :]$ javadoc -d MaDoc HelloWorld.java
Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=UTF-8
Loading source file HelloWorld.java...
...
Generating MaDoc/help-doc.html...
[10:19][Prog pc666 :]$

```

Commenter :

- ▶ chaque classe,
- ▶ chaque champ,
- ▶ chaque méthode.

**Pour bien restituer les caractères accentués (GNU-LINUX)**

Au sein du fichier .bashrc (à la racine de votre compte), on positionnera correctement la variable du shell BASH nommée `JAVA_TOOL_OPTIONS` (cf. première ligne du terminal ci-dessus).

# La documentation – exemple avec JAVADOC

```

1  /**
2   * @author O.Marchetti (mon e-mail)
3   * @version 1.0
4   */
5  public class HelloWorld {
6      /**
7       * Cette méthode est le point d'entrée du
8       * programme.
9       *
10      * <p> Ici, on mettra un commentaire plus
11      * long si nécessaire. </p>
12      *
13      * @param args est un tableau de type
14      * String désignant les chaînes de
15      * caractères présentes sur la ligne de
16      * commande.
17      *
18      * @return un entier indiquant que tout
19      * s'est bien passé.
20      */
21      public static int main(String args[]) {
22          System.out.println("Hello world !");
23          return 0;
24      }
25  }

```

```

[10:19][Prog pc666 :]$ grep JAVA ~/.bashrc
export JAVA_TOOL_OPTIONS=-Dfile.encoding=UTF-8
[10:19][Prog pc666 :]$ javadoc -d MaDoc HelloWorld.java
Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=UTF-8
Loading source file HelloWorld.java...
...
Generating MaDoc/help-doc.html...
[10:19][Prog pc666 :]$ ls

```

Commenter :

- ▶ chaque classe,
- ▶ chaque champ,
- ▶ chaque méthode.

**Pour bien restituer les caractères accentués (GNU-LINUX)**

Au sein du fichier .bashrc (à la racine de votre compte), on positionnera correctement la variable du shell BASH nommée `JAVA_TOOL_OPTIONS` (cf. première ligne du terminal ci-dessus).

# La documentation – exemple avec JAVADOC

```

1  /**
2   * @author O.Marchetti (mon e-mail)
3   * @version 1.0
4   */
5  public class HelloWorld {
6      /**
7       * Cette méthode est le point d'entrée du
8       * programme.
9       *
10      * <p> Ici, on mettra un commentaire plus
11      * long si nécessaire. </p>
12      *
13      * @param args est un tableau de type
14      * String désignant les chaînes de
15      * caractères présentes sur la ligne de
16      * commande.
17      *
18      * @return un entier indiquant que tout
19      * s'est bien passé.
20      */
21      public static int main(String args[]) {
22          System.out.println("Hello world !");
23          return 0;
24      }
25  }

```

```

[10:19][Prog pc666 :]$ grep JAVA ~/.bashrc
export JAVA_TOOL_OPTIONS=-Dfile.encoding=UTF-8
[10:19][Prog pc666 :]$ javadoc -d MaDoc HelloWorld.java
Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=UTF-8
Loading source file HelloWorld.java...
...
Generating MaDoc/help-doc.html...
[10:19][Prog pc666 :]$ ls
HelloWorld.java  MaDoc
[10:20][Prog pc666 :]$

```

Commenter :

- ▶ chaque classe,
- ▶ chaque champ,
- ▶ chaque méthode.

**Pour bien restituer les caractères accentués (GNU-LINUX)**

Au sein du fichier .bashrc (à la racine de votre compte), on positionnera correctement la variable du shell BASH nommée `JAVA_TOOL_OPTIONS` (cf. première ligne du terminal ci-dessus).

# La documentation – exemple avec JAVADOC

```

1  /**
2   * @author O.Marchetti (mon e-mail)
3   * @version 1.0
4   */
5  public class HelloWorld {
6      /**
7       * Cette méthode est le point d'entrée du
8       * programme.
9       *
10      * <p> Ici, on mettra un commentaire plus
11      * long si nécessaire. </p>
12      *
13      * @param args est un tableau de type
14      * String désignant les chaînes de
15      * caractères présentes sur la ligne de
16      * commande.
17      *
18      * @return un entier indiquant que tout
19      * s'est bien passé.
20      */
21      public static int main(String args[]) {
22          System.out.println("Hello world !");
23          return 0;
24      }
25  }

```

```

[10:19][Prog pc666 :]$ grep JAVA ~/.bashrc
export JAVA_TOOL_OPTIONS=-Dfile.encoding=UTF-8
[10:19][Prog pc666 :]$ javadoc -d MaDoc HelloWorld.java
Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=UTF-8
Loading source file HelloWorld.java...
...
Generating MaDoc/help-doc.html...
[10:19][Prog pc666 :]$ ls
HelloWorld.java  MaDoc
[10:20][Prog pc666 :]$ ls MaDoc/

```

Commenter :

- ▶ chaque classe,
- ▶ chaque champ,
- ▶ chaque méthode.

**Pour bien restituer les caractères accentués (GNU-LINUX)**

Au sein du fichier .bashrc (à la racine de votre compte), on positionnera correctement la variable du shell BASH nommée `JAVA_TOOL_OPTIONS` (cf. première ligne du terminal ci-dessus).

# La documentation – exemple avec JAVADOC

```

1  /**
2   * @author O.Marchetti (mon e-mail)
3   * @version 1.0
4   */
5  public class HelloWorld {
6      /**
7       * Cette méthode est le point d'entrée du
8       * programme.
9       *
10      * <p> Ici, on mettra un commentaire plus
11      * long si nécessaire. </p>
12      *
13      * @param args est un tableau de type
14      * String désignant les chaînes de
15      * caractères présentes sur la ligne de
16      * commande.
17      *
18      * @return un entier indiquant que tout
19      * s'est bien passé.
20      */
21      public static int main(String args[]) {
22          System.out.println("Hello world !");
23          return 0;
24      }
25  }

```

```

[10:19][Prog pc666 :]$ grep JAVA ~/.bashrc
export JAVA_TOOL_OPTIONS=-Dfile.encoding=UTF-8
[10:19][Prog pc666 :]$ javadoc -d MaDoc HelloWorld.java
Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=UTF-8
Loading source file HelloWorld.java...
...
Generating MaDoc/help-doc.html...
[10:19][Prog pc666 :]$ ls
HelloWorld.java  MaDoc
[10:20][Prog pc666 :]$ ls MaDoc/
HelloWorld.html      help-doc.html      package-list
...
deprecated-list.html  package-frame.html stylesheet.css
[10:20][Prog pc666 :]$

```

Commenter :

- ▶ chaque classe,
- ▶ chaque champ,
- ▶ chaque méthode.

**Pour bien restituer les caractères accentués (GNU-LINUX)**

Au sein du fichier .bashrc (à la racine de votre compte), on positionnera correctement la variable du shell BASH nommée `JAVA_TOOL_OPTIONS` (cf. première ligne du terminal ci-dessus).

# La documentation – exemple avec JAVADOC

```

1  /**
2   * @author O.Marchetti (mon e-mail)
3   * @version 1.0
4   */
5  public class HelloWorld {
6      /**
7       * Cette méthode est le point d'entrée du
8       * programme.
9       *
10      * <p> Ici, on mettra un commentaire plus
11      * long si nécessaire. </p>
12      *
13      * @param args est un tableau de type
14      * String désignant les chaînes de
15      * caractères présentes sur la ligne de
16      * commande.
17      *
18      * @return un entier indiquant que tout
19      * s'est bien passé.
20      */
21      public static int main(String args[]) {
22          System.out.println("Hello world !");
23          return 0;
24      }
25  }

```

```

[10:19][Prog pc666 :]$ grep JAVA ~/.bashrc
export JAVA_TOOL_OPTIONS=-Dfile.encoding=UTF-8
[10:19][Prog pc666 :]$ javadoc -d MaDoc HelloWorld.java
Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=UTF-8
Loading source file HelloWorld.java...
...
Generating MaDoc/help-doc.html...
[10:19][Prog pc666 :]$ ls
HelloWorld.java  MaDoc
[10:20][Prog pc666 :]$ ls MaDoc/
HelloWorld.html      help-doc.html      package-list
...
deprecated-list.html  package-frame.html stylesheet.css
[10:20][Prog pc666 :]$ firefox MaDoc/index.html &

```

Commenter :

- ▶ chaque classe,
- ▶ chaque champ,
- ▶ chaque méthode.

**Pour bien restituer les caractères accentués (GNU-LINUX)**

Au sein du fichier .bashrc (à la racine de votre compte), on positionnera correctement la variable du shell BASH nommée `JAVA_TOOL_OPTIONS` (cf. première ligne du terminal ci-dessus).

# La documentation – exemple avec JAVADOC – résultat

PACKAGE **CLASS** TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY NESTED FIELD CONSTR METHOD DETAIL FIELD CONSTR METHOD

## Class HelloWorld

java.lang.Object  
HelloWorld

---

public class HelloWorld  
extends java.lang.Object

### Constructor Summary

**Constructors**

Constructor and Description

HelloWorld()

### Method Summary

**All Methods** Static Methods Concrete Methods

Modifier and Type	Method and Description
static int	main(java.lang.String[] args) Cette méthode est le point d'entrée du programme.

Methods Inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

### Constructor Detail

HelloWorld

public HelloWorld()

### Method Detail

**main**

public static int main(java.lang.String[] args)

Cette méthode est le point d'entrée du programme.

Ici, on mettra un commentaire plus long si nécessaire.

Parameters:

args - est un tableau de type String désignant les chaînes de caractères présentes sur la ligne de commande.

Returns:

un entier indiquant que tout s'est bien passé

PACKAGE **CLASS** TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY NESTED FIELD CONSTR METHOD DETAIL FIELD CONSTR METHOD

Documenter un programme est simple :

- ▶ avec les commentaires ordinaires (minimum syndical),
- ▶ avec les commentaires JAVADOC.

Documenter est une tâche essentielle :

- ▶ un code mal-documenté est difficile à
  - prendre en main,
  - entretenir,
  - partager.
- ▶ un code non-documenté risque de finir à la poubelle.