Module de mise à niveau en Java
Cours 1

Laure KAHLEM
laure.kahlem@univ-orleans.fr

Références

- Mickaël Kerboeuf. Fondements de la programmation orientée objet avec Java8. Ellipses, 2016.
- Kathy Sierra et Bert Bates. Java Tête la première. O'Reilly, 2007.
- David Flanagan. Java in a Nutshell. Manuel de référence. O'Reilly, 2002.
- Cyrille Herby. Apprenez à programmer en JAVA. Broché, 2012.
- José Paumard. « Java en ligne » [en ligne]. http://blog.paumard.org/cours/java/ (Consulté le 15/07/2015).

Références

- Tutos d'Oracle : https://docs.oracle.com/javase/tutorial/
- Thinking in Java (http://mindview.net/Books/TIJ/) / Penser en Java (http://penserenjava.free.fr/)
- Livre : Core Java, Prentice-Hall (vol. 1) / Au coeur de Java, Campus Press
- Le doudoux (http://www.jmdoudoux.fr/, 3413 pages)
- Internet :
  - developpez.com
  - dzone.com [refcardz]
  - slides de Richard Grin, Miage Nice
  - les cast codeurs
  - java code geek, java ranch, javaworld, InfoQ, … Stack Overflow !
  - Javarevisited (http://javarevisited.blogspot.fr)
  - twitter / youtube (devoxx FR)

Le langage Java

- **Langage de programmation orienté objet**
- **Portable sur différents systèmes d'exploitation grâce à une machine virtuelle (JVM)**
  - **Pour exécuter un programme Java**
    - Installer le **JRE (J**ava **R**untime **E**nvironnement**)**
  - **Pour développer en Java**
    - Installer le **JDK (J**ava **D**evelopment **K**it**)**
    - Version conseillée **Java SE 17** (Java Standart Edition)

Le langage Java

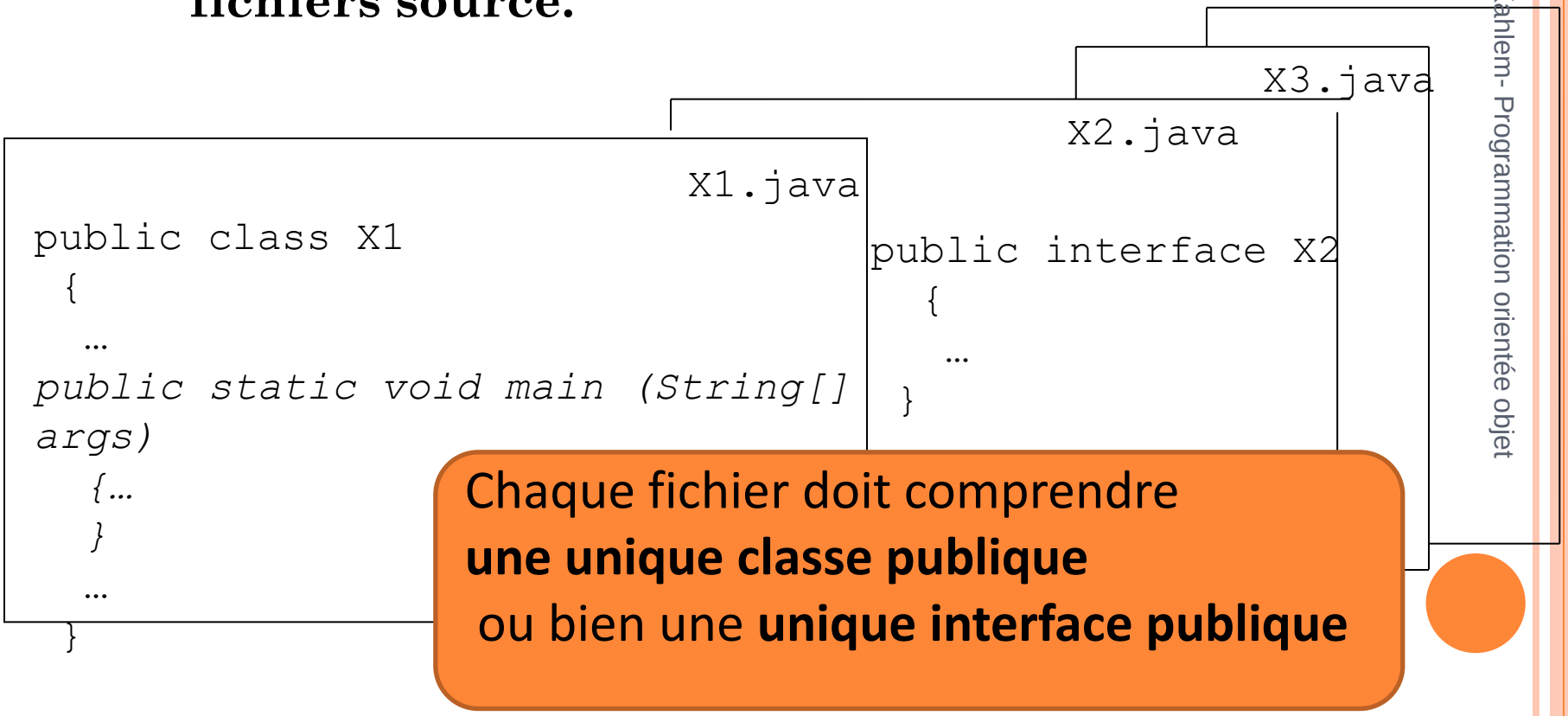
- **Java dispose d'une API (interface de programmation applicative) très riche (multimédia, interfaces graphiques, connectivité réseau, base de données…)**
  - Nombreuses classes structurées **en packages**
  - Consulter les versions postérieures à Java 8.
  - http://docs.oracle.com/javase/8/docs/api
- **Java est sûr**
  - Langage fortement typé
  - De nombreuses vérifications lors de l'exécution garantissent son bon fonctionnement

Le langage Java

- **Plusieurs plateformes de développement sont proposées par Oracle suivant les types d'applications**
  - **Java SE :** Java Platform, **Standard Edition**
  - **Java EE** : **Enterprise Edition**, ajoute à Java SE les API pour écrire des applications n-tiers: servlets, JSP, JSF, EJB,…
  - **Java ME : Micro Edition**, ajoute à Java SE des outils dédiés au développement d'applications mobiles embarquées sur des terminaux légers (carte à puce/Java card, téléphone portable,…).

COMMENT FONCTIONNE JAVA?

- **Construction d'un programme JAVA :**
  - Un programme Java est composé **d'un ensemble de fichiers source.**

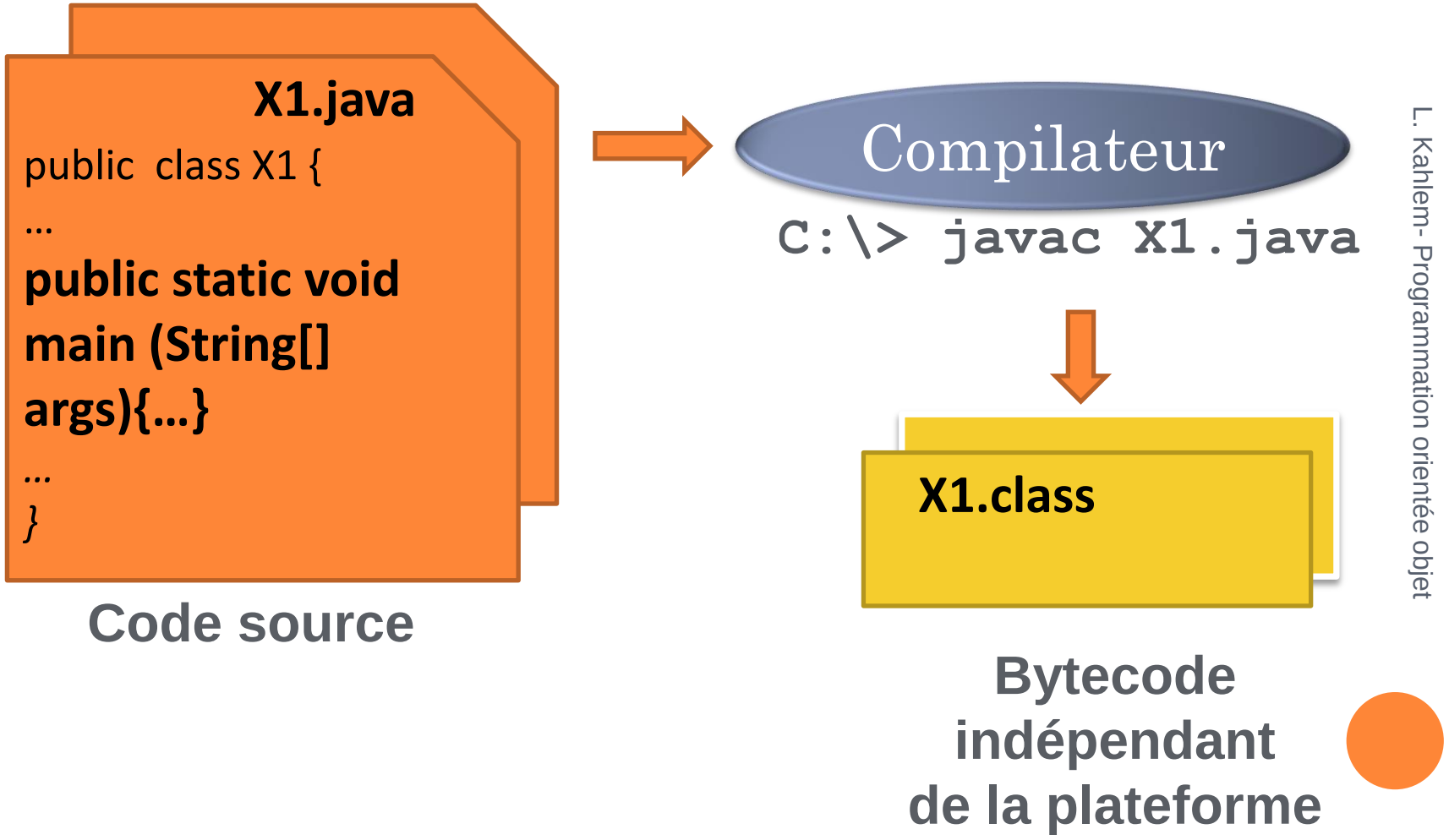


Comment fonctionne Java?

- **Compilation et exécution d'un programme JAVA**
  - Le code source doit être traduit en langage machine.
  - Deux étapes en Java:
    1. Le compilateur traduit le **code source** dans un langage intermédiaire , le **bytecode** .
       *Compilateur fourni par le JDK : **javac***
    2. Le bytecode est traduit dans le langage natif du processeur de l'ordinateur par la **JVM**.
       *Machine virtuelle fourni par le JDK : **java***

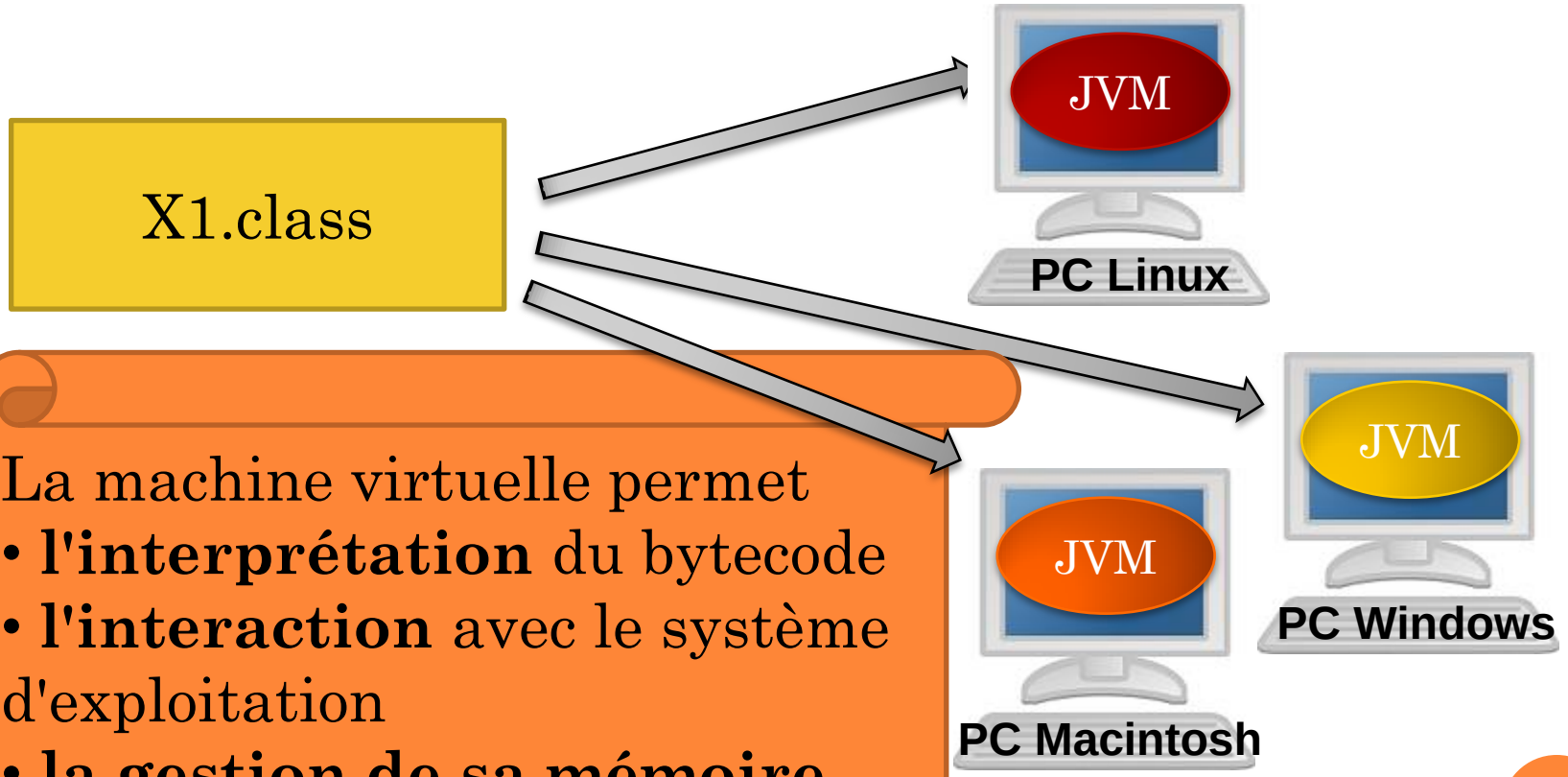
Comment fonctionne Java?

- **Compilation et exécution d'un programme JAVA**



Comment fonctionne Java?

- **Compilation et exécution d'un programme JAVA**



La machine virtuelle permet
- **l'interprétation** du bytecode
- **l'interaction** avec le système d'exploitation
- **la gestion de sa mémoire** grâce au ramasse miettes

```
Exécution :
C:\> java X1
```

Comment fonctionne Java?

- **Compilation et exécution d'un programme JAVA**
  - Durant l'exécution d'un programme **les classes sont chargées dynamiquement dans la JVM** en fonction des besoins.
  - Une classe peut être chargée
    - depuis une machine locale
    - depuis une autre machine par le réseau
    - par tout autre moyen (base de données…)

Environnement de travail

- Plusieurs IDE :
  - **Eclipse IDE** for Java Developpers http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/oxygen3a
  - **IntelliJ** https://www.jetbrains.com/idea/
    - version gratuite [Community] et payante [Ultimate], licences accessibles dans les salles de TP de la fac
    - pour chez vous : https://www.jetbrains.com/shop/eform/students inscription avec votre mail de l'Université
- En TP, nous utiliserons **IntelliJ.**

Les structures fondamentales de Java

- **Les commentaires**
  - **Sur une ligne :** `//`
    - `int nb = 0; // nb représente le nombre d'étudiants`
  - **Sur plusieurs lignes :** `/* … */`
- **Commentaires JavaDoc**
  - Commencent par : `/**`
  - Se terminent par : `*/`
  - Les tags spécifiques sont introduits par :@
    - @author, @param, @return, @see, @throws
  - http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html (consulté le 04/09/2021)

Les structures fondamentales de Java

- **Les annotations**
  - **Instructions destinées au compilateur**
  - **Mots-clés précédé de @**
  - **Exemples**
    - **Déclaration et redéfinition de méthode**

      `@Override`
    - **Déclaration d'une interface fonctionnelle**

      `@FunctionalInterface`
    - **Suppression des avertissements de compilation**

      `@SuppressWarning`
    - **Annotation des méthodes dans Junit 4**

      `@Test, @Before, @After, @BeforeClass, @AfterClass`

Les structures fondamentales de Java

- **Les identifiants**
  - **La première lettre d'un identifiant est**
    - Soit un caractère alphabétique non accentué
    - Soit $ ou _.
  - **Les autres lettres peuvent être** des caractères parmi les chiffres, les lettres et même les caractères unicode.

    Les lettres accentuées et caractères unicode sont déconseillés.
  - **Un identifiant ne doit pas être un mot réservé.**

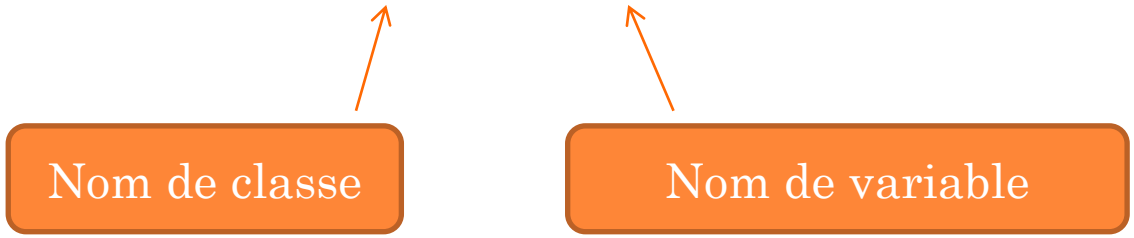
Les structures fondamentales de Java

- **Mots réservés**

abstract	assert	boolean	break	byte	case
catch	char	class	const	continue	default
do	double	else	enum	extends	false
final	finally	float	for	goto	if
implements	import	instanceof	int	interface	long
native	new	null	package	private	protected
public	return	short	static	strictfp	super
switch	this	synchronized	throw	throws	transient
true	try	void	volatile	while	

Les structures fondamentales de Java

- **Choix d'un identifiant**
  - **Conventions de nommage**
    - **`Abc`** : nom de classe, d'interface ou de type énuméré
    - **`ABC`** : nom de constante
    - **`abcDef`**: nom de variable, d'attribut, de paramètre, de méthode, de package.
    - Exemple: `String nomEtudiant;`



Les structures fondamentales de Java

- **Les types de données**
  - **Les types primitifs**
  - **Les classes dont les objets sont les instances**

Les structures fondamentales de Java

- **Les types primitifs**
  - Les types primitifs se manipulent en Java uniquement **par valeur.**
  - Les valeurs que représentent le type primitif peuvent être :
    - stockées dans **des variables.**
    - ou nommées directement par des littéraux spécifiques du langage.

  Exemple :
  - 56 est un littéral du type primitif `int` qui représente la *valeur* 56.

  Cette valeur peut être stockée dans une variable : `int x;`

  `x = 56;`

L'affectation est une expression qui met à jour la variable et retourne la valeur affectée.

Les structures fondamentales de Java

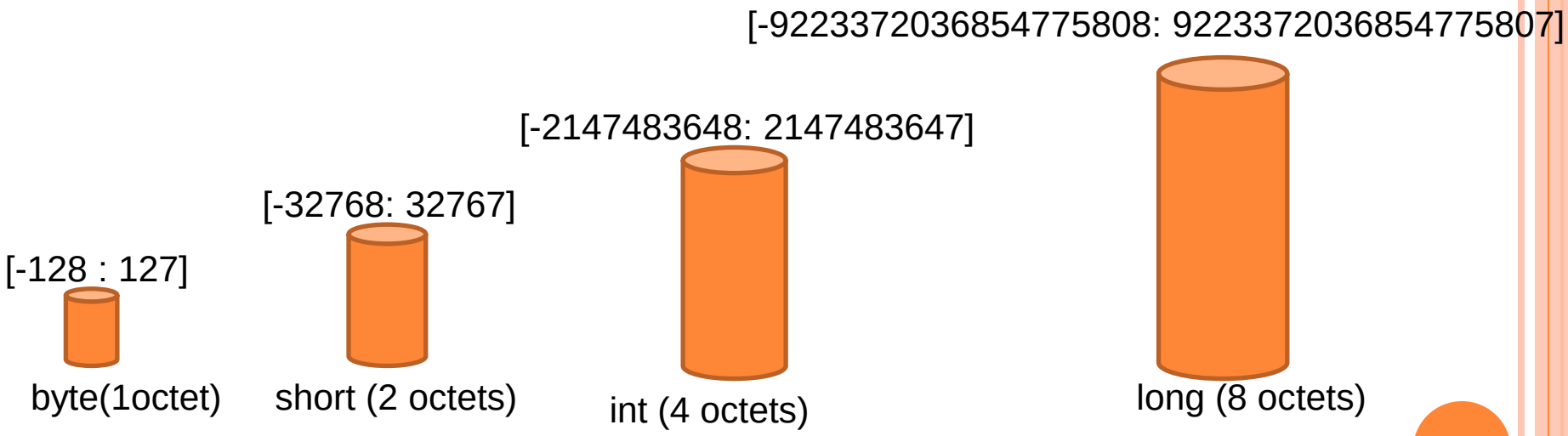
- **Les types primitifs**
  - **Type booléen :** `boolean`
    - Deux littéraux : **`true`** et **`false`**.
  - **`Type caractère: char`** (Java utilise l'encodage **unicode** sur 2 octets)
    - `Littéraux délimités par '.`
    - `Caractères spéciaux introduits par \.`

Exemples :

Caractère **u** : 'u'
Caractère **4** : '4'
Caractère \ : '\\'
Caractère ' : '\' '
Caractère @ de code unicode 0040: '\u0040'

Les structures fondamentales de Java

- **Types primitifs**
  - **Types numériques entiers**
    - *byte, short, int, long*



Les structures fondamentales de Java

- **Types primitifs**
  - **Types numériques entiers**
    - **Littéraux de type byte, short, int :** valeurs entières signées ou non.
    - **Les littéraux de type long :** valeurs entières signées ou non suffixées par le caractère L ou l.
    - **Exemple :**
      - `byte b = -5;`
      - `int j = 2000000000;`
      - `long = 12349L;`
      - `int k = -0b11011011;`

Écriture binaire d'un littéral représentant la valeur -219

Les structures fondamentales de Java

- **Types primitifs**
  - **Types numériques flottants**

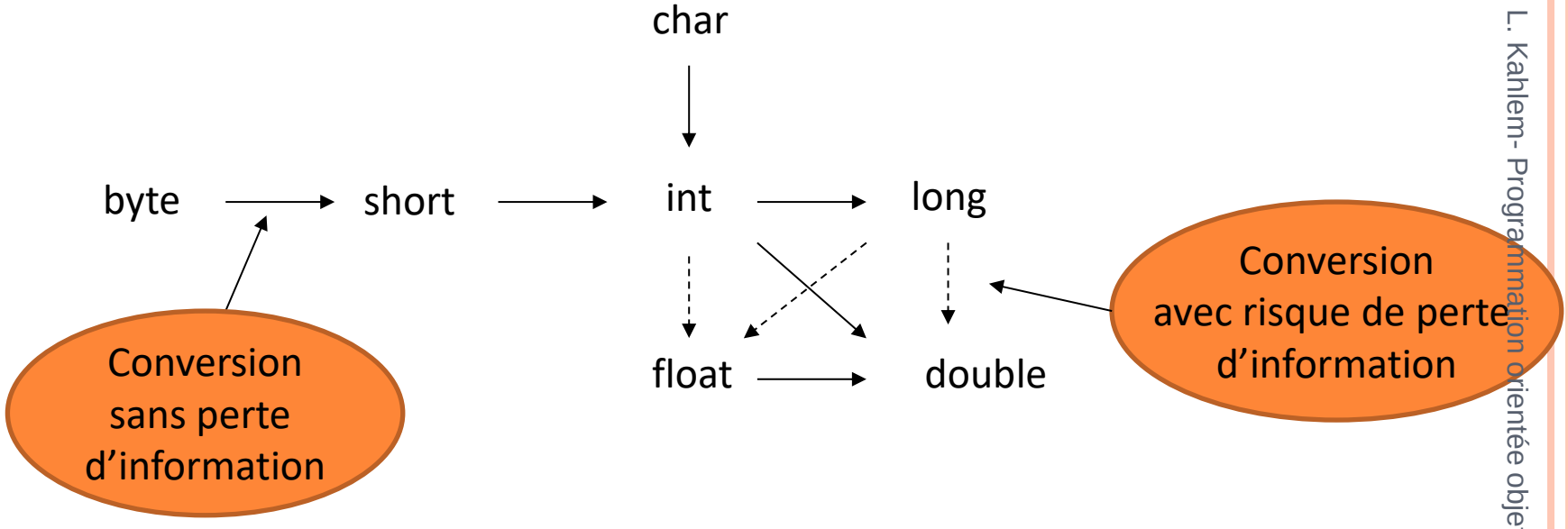
Types	**float**	**double**
Plus petite valeur absolue	$1,4.10^{-45}$	$4,9.10^{-324}$
Plus grande valeur absolue	$3,4028235.10^{38}$	$1,79769313486231 57.10^{308}$
Littéral correspondant la valeur : $1,456.10^{2}$ (=145,6)	1.456E2**f**	1.456E2 ou1.456E2d
Littéral correspondant la valeur : $-34,1.10^{-3}$ (=-0,0341)	-34.1E3**f**	-34.1E3 ou-34.1E3d

Les structures fondamentales de Java

- **Types primitifs**
  - **Types numériques flottants**
    - Les types `float` et `double` ne sont pas précis.
    - Il ne faut pas les utiliser pour des calculs financiers . Dans ce cas, utiliser la classe `BigDecimal`.
    - Les types `float` et `double` incluent des valeurs spéciales `NaN` (Not a Number) et (+/-)`Infinity`.
      Exemple : `2.0/0=>Infinity`

Les structures fondamentales de Java

- **Types primitifs**
  - **Conversions légales de types numériques**



Exemple :

```
int n= 123456789;
float f = n; //f vaut 1.23456792E8
int i = 'L'; //i vaut 76 (encodage unicode)
```

Les structures fondamentales de Java

- **Types primitifs**
  - **Transtypage (`cast`):** Permet de convertir un type plus grand en un type plus petit. Cette conversion pouvant entraîner des pertes d'information.

    Exemples :

```
double d = 5.6545;
int i = (int)d; // i = 5
```


Les structures fondamentales de Java

- **Types primitifs**
  - **Opérateurs usuels sur les types numériques:**
    - `= + - * / % ++ -- += -= *= /=`
    - `== != < > <= >=`
    - `& || !`
  - **`x++`** : la valeur actuelle de x est utilisée dans l'expression puis ensuite x est incrémentée.
  - **`++x`** : la valeur actuelle de x est incrémentée puis ensuite elle est utilisée dans l'expression.

Les structures fondamentales de Java

- **Types primitifs**
  - **Promotion numérique**
    - Tout calcul entre entiers (byte, short, int, long) donne un résultat de type `int` ou `long` (cas où au moins un des opérandes est de type `long`).
    - Tout calcul entre une opérande flottante et une entière donne un résultat de type flottant.
    - Tout calcul entre deux opérandes flottantes donne un résultat du type flottant le plus large.

Les structures fondamentales de Java

- **Exercice 1: Évaluation d'expressions arithmétiques**

On considère les déclarations suivantes :

```
byte b1 = 5, b2 = 30 ;
short s = 300 ;
int n = 600 ;
long q = 1500 ;
float x = 1.5E2f ;
double y = 5.0E3 ;
```

Donner le type des expressions arithmétiques suivantes :

```
b1 + b2
s + b1
q + s * (b1 + b2)
x + q * n
b1 * q / x
b1 * q * 2./ x
b1 * q * 2.f/ y
```

Les structures fondamentales de Java

- **Exercice 2 : Conversion automatique du type char vers le type int**

On considère les déclarations suivantes :

```
char c = 50, ci = 'i', ck = 'k' ;
byte b = 20 ;
```

Donner le type et la valeur des expressions arithmétiques suivantes :

```
c + 1
2 * c
ck - ci
b * c
```

Les structures fondamentales de Java

- **Exercice 3 :**

On considère les déclarations suivantes :

```
int nbre1 = 3, nbre2 = 2;
```

Donner la valeur de la variable `resultat`:

```
double resultat= nbre1/nbre2;
```


Les structures fondamentales de Java

- **Les classes et leurs instances**
  - **Une classe** définit un **type**.
  - Les valeurs associées sont des **objets**, instances de la classe.

Définition du corps de la classe : `class A {…}`

Instanciation de la classe A : `new A()`

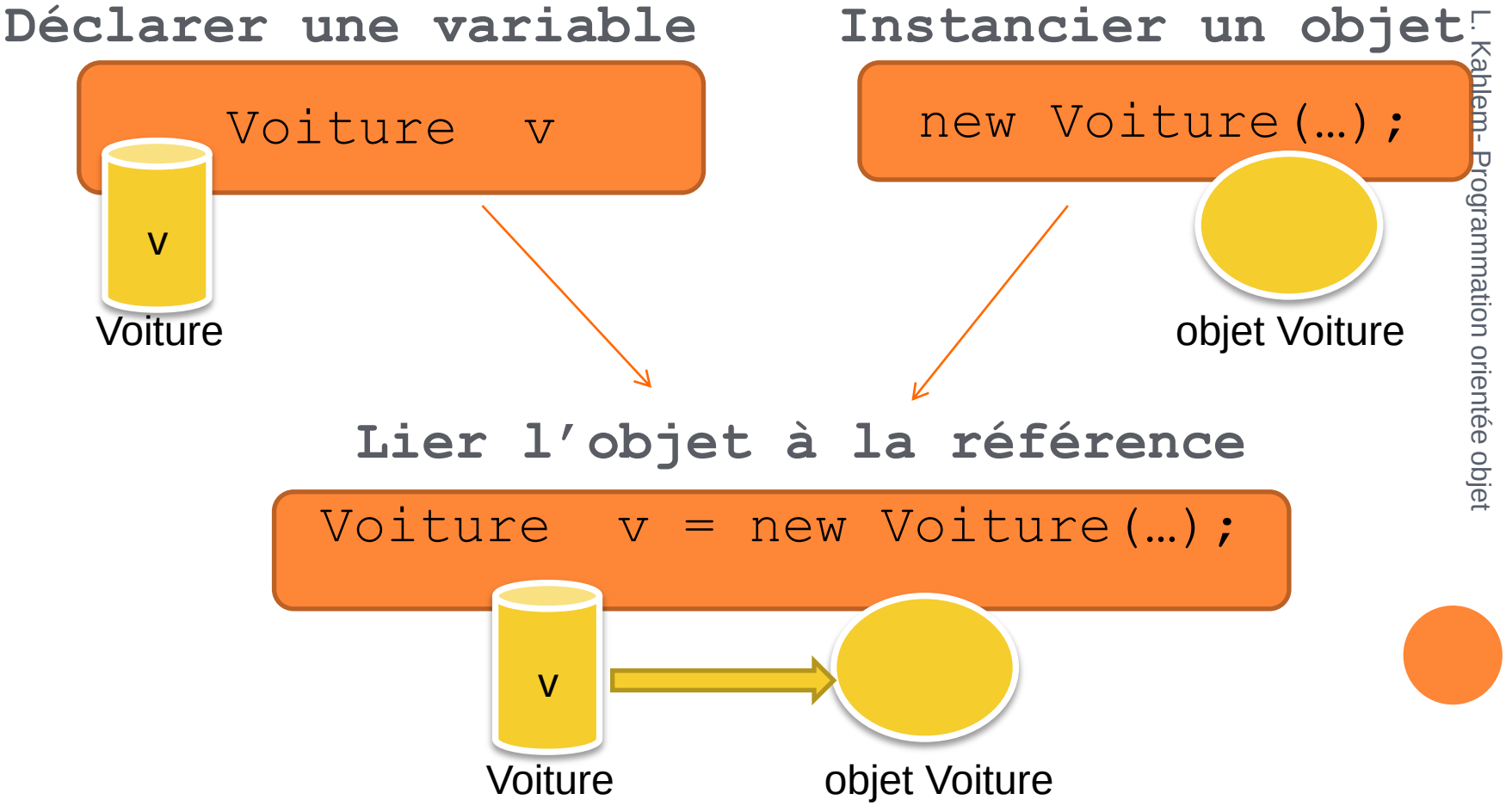
Allocation dynamique par l'opérateur new

Les structures fondamentales de Java

- **Les classes et leurs instances**
  - Les objets sont manipulables **par référence** et non par valeur
  - **Les variables** de type objet contiennent **l'adresse mémoire** qui permet d'accéder à l'objet.
  - Valeur particulière qui ne désigne aucun objet: **`null`**.

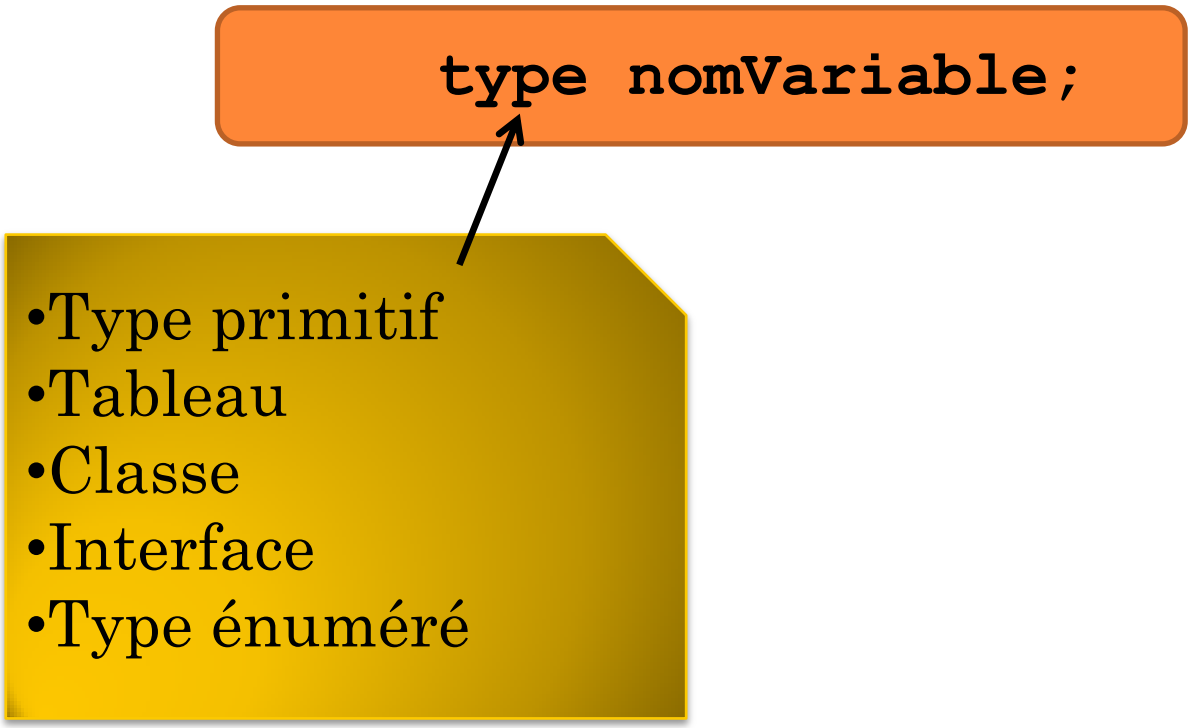
Les structures fondamentales de Java

- **Les classes et leurs instances**



Les structures fondamentales de Java

- **Les variables**
  - Déclaration de variable



Les structures fondamentales de Java

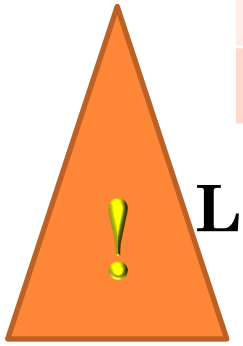
- **Les variables**
  - Une variable a une portée limitée au bloc de code dans lequel elle est déclarée (délimité par `{ . . . }`).
  - Une variable peut être déclarée au niveau d'une classe : **attribut** de la classe (portée peut être étendue à l'extérieur de la classe).
  - Dans tous les autres cas, une variable est **locale**.
  - **Valeur affectée à une variable:**
    - Littéral
    - Contenu d'une autre variable
    - Valeur d'une expression
    - Résultat de l'appel de fonction.

Les structures fondamentales de Java

- **Les variables**

  Les attributs sont initialisés par défaut

Type	Valeur par défaut
`byte, short, int, long`	`0`
`float, double`	`0.0`
`boolean`	`false`
`char`	`'\u0000'`
`objet`	`null`



Les variables locales doivent être obligatoirement initialisées.

Les structures fondamentales de Java

- **La pile et le tas : deux espaces mémoire distincts**
  - L'espace mémoire alloué à une **variable locale lors de l'exécution d'une fonction ou d'un bloc de code** est situé **sur la pile.**
  - **La création d'un objet** avec l'opérateur **`new`** entraîne l'allocation dynamique d'un espace **sur le tas**.
  - Si une variable contient une référence à un objet, la valeur de cette référence est stockée dans la pile.
  - Si un objet n'est plus référencé, c'est le **ramasse-miettes** (garbage collector) qui se charge de libérer l'espace mémoire alloué.

Les structures fondamentales de Java

- **Instructions conditionnelles**
  - `if (condition) instruction`
  - `if (condition)`
    `  instruction1`
    `else`
    `  instruction2`

Les structures fondamentales de JAVA

- **Branchements conditionnels à choix multiples:**

```
if (condition1)
 instruction1
else if (condition2)
 instruction2
…
else if (condition3)
 instruction3
else
 instruction par défaut
```

Les structures fondamentales de Java

- **Branchements conditionnels à choix multiples:** le sélecteur `switch`
  - Type de la valeur énumérée pour sélectionner
    - Type primitif entier
    - Type primitif `char`
    - Chaîne de caractères : `String`
    - Objet de type `Enum`
  - L'instruction `break` en fin de clause `case` permet de sortir du `switch`. Sans break, le code de la clause suivante s'exécute en séquence.

Les structures fondamentales de Java

- **Branchements conditionnels à choix multiples:** le sélecteur `switch`
- Exemple : Soit un caractère `c`

```
switch (c) {
    case '-':
      sign=-1;
    case '+':
      c= getChar();
      break;
    case '.':
      break;
    default:
      if (!isdigit(c))
            return 0;
      break;
}
```

```
if (c == '-'){
    sign=-1;
    c= getChar();
}
else if (c == '+'){
    c= getChar();
}
else if (c != '.' && !isdigit(c)) {
    return 0;
}
```

Les structures fondamentales de Java

- **Instructions conditionnelles**
  - Opérateur conditionnel **`?:`** :

    l'expression ***`condition ? expression1 : expression2`*** est évaluée à **`expression1`** si la condition est vraie, à **`expression2`** sinon.

    Exemple : `min = (a < b )? a : b`

Les structures fondamentales de Java

- **Instructions itératives**
  - Boucles `while`
    - `while (condition) instruction;`
    - `do` instruction `while (condition);`

Les structures fondamentales de Java

- **Instructions itératives**
  - Boucles `for`
    - **`for ( expr1; expr2; expr3 ) boucle`**
      - `expr1` est l'expression d'initialisation,
      - `expr2` est la condition de continuation, si sa valeur booléenne est *false*, sortie de la boucle.
      - `expr3` est l'expression d'incrémentation.
    - **`for ( type variable : collection ) boucle`**

      Boucle de type `for each` permet de parcourir séquentiellement les éléments d'une collection.

Les structures fondamentales de Java

- **Tableaux à une dimension**
  - Un tableau est une collection de **valeurs indicées** (à partir de 0).

Un tableau est un **objet**

Une variable de type tableau **contient une référence**

Un tableau est **alloué dynamiquement** avec l'opérateur **new**

Les structures fondamentales de Java

- **Tableaux à une dimension**

Un tableau est un **objet**

Un tableau dispose d'un **attribut** qui contient sa taille : `length`

Les cases du tableau sont assimilées aux **attributs de l'objet**

Les cases du tableau ont **des valeurs initiales par défaut.**

Les structures fondamentales de Java

- **Tableaux à une dimension**
  - Exemple de déclaration d'une variable **`tab1`** de type tableau d'entiers:
    - **`int[] tab1;`**
  - Exemple de création du tableau d'entiers
    - **`int[] tab1 = new int[5];`**
  - Initialisation rapide d'un tableau d'entiers (pas besoin d'appeler *new*)
    - **`int[] tab1 = {23,45,6,7};`**

La dimension du tableau est fournie au moment de sa création.

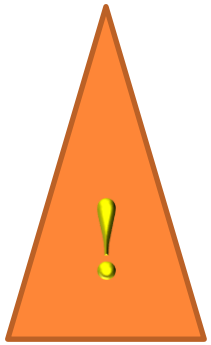
Les structures fondamentales de Java

- **Tableaux à une dimension**
  - **Parcours du tableau tab1:**
    - ```
      for ( int i  =  0; i < tab1.length; i++)
          System.out.println(tab1[i]);
      ```
    - ```
      for ( int element : tab1)
          System.out.println( element );
      ```

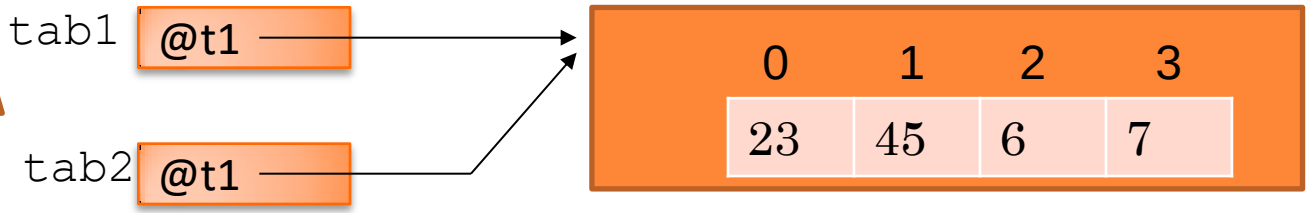

Les structures fondamentales de Java

- **Tableaux**
  - Copie d'une variable tableau dans une autre

```
int[] tab1 = {23, 45, 6, 7};
int[] tab2 = tab1;
```



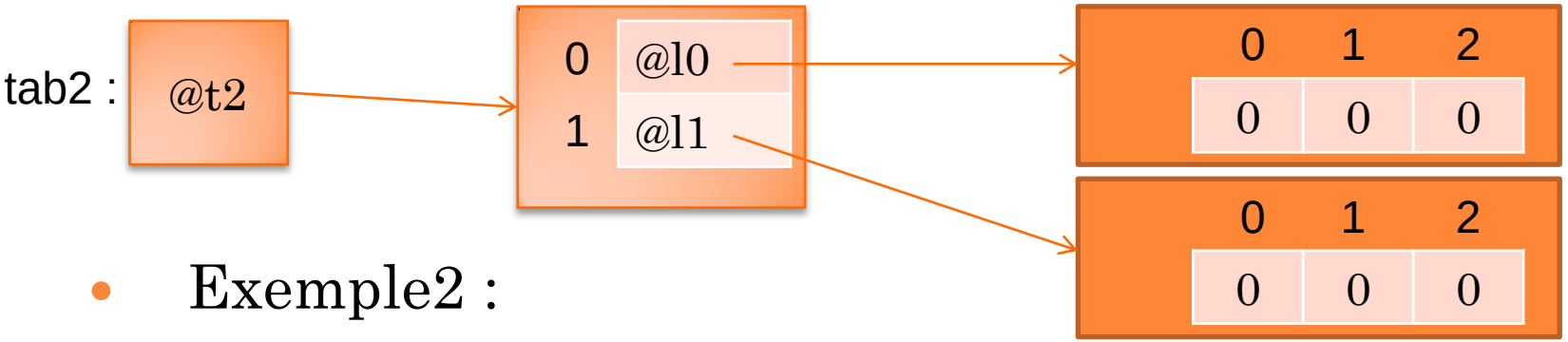
tab1 et tab2 font référence au même objet tableau.



Les structures fondamentales de Java

- **Tableaux à 2 dimensions**
  - **Un tableau à 2 dimensions est un tableau à 1 dimension de tableaux à 1 dimension**
  - Exemple1 :

```
int[][] tab2 = new int [2][3];
```



  - Exemple2 :

```
int[][] tab3 = new int[2][];
```