

Cours Java Standard

Présenté par : M^r Djamel MOUCHENE

24/08/2021

Un peu d'histoire [1 / 2]

- ▶ 1990 : Sun Microsystems – *James Gosling fondateur du langage.*

Un peu d'histoire [2/2]

- ▶ Indépendant de toute architecture système → Idéal pour programmer des applications utilisables dans des réseaux hétérogènes : Internet.
- ▶ Java devint alors un enjeu stratégique pour Sun : HotJava
- ▶ La version 2.0 du navigateur de Netscape a été développée pour supporter Java,
- ▶ Suivi de près par Microsoft (Internet Explorer 3)
- ▶ L'intérêt pour la technologie Java s'est accru rapidement: IBM, Oracle et d'autres ont pris des licences Java.

Les différentes versions de Java

► De nombreuses versions de Java depuis 1995

Version	Release date	End of Free Public Updates ^{[5][6]}	Extended Support Until
JDK Beta	1995	<p>Java 1.2 en 1999 → Java 2, version 1.2</p> <p>Java 1.3 en 2001 → Java 3, version 1.3</p> <p>Java 1.4 en 2002 → Java 4, version 1.4</p>	
JDK 1.0	January 1996		
JDK 1.1	February 1997		
J2SE 1.2	December 1998		
J2SE 1.3	May 2000		
J2SE 1.4	February 2002	October 2008	February 2013
J2SE 5.0	September 2004	November 2009	April 2015
Java SE 6	December 2006	April 2013	December 2018
Java SE 7	July 2011	April 2015	July 2022

Les différentes versions de Java

Version	Release date	End of Free Public Updates ^{[5][6]}	Extended Support Until
Java SE 8 (LTS)	March 2014	January 2019 for Oracle (commercial) December 2020 for Oracle (personal use) At least May 2026 for AdoptOpenJDK At least June 2023 ^[7] for Amazon Corretto	December 2030
Java SE 9	September 2017	March 2018 for OpenJDK	N/A
Java SE 10	March 2018	September 2018 for OpenJDK	N/A
Java SE 11 (LTS)	September 2018	At least August 2024 ^[7] for Amazon Corretto October 2024 for AdoptOpenJDK	September 2026
Java SE 12	March 2019	September 2019 for OpenJDK	N/A
Java SE 13	September 2019	March 2020 for OpenJDK	N/A
Java SE 14	March 2020	September 2020 for OpenJDK	N/A

Les différentes versions de Java

Version	Release date	End of Free Public Updates ^{[5][6]}	Extended Support Until
Java SE 14	March 2020	September 2020 for OpenJDK	N/A
Java SE 15	September 2020	March 2021 for OpenJDK	N/A
Java SE 16	March 2021	September 2021 for OpenJDK	N/A
Java SE 17 (LTS)	September 2021	TBA	TBA

Analyse du problème [1 / 2]

► Se poser les bonnes questions

- Quelles sont les **éléments** qui interviennent dans le problème ?
- Quelles sont les **données**, les **objets**, que le programme va manipuler !?
- Quelles vont être les **relations** entre ces objets ?
- Quelles sont les **opérations** que je vais pouvoir effectuer sur ces objets?

Analyse du problème [2 / 2]

► Savoir être :

- **Efficace** : Simple, clair, *rapide* !!??
- **Paresseux** : Réutilisation du code ?
- **Prévoyant** : Facilement réutilisable et extensible ?

Caractéristiques du langage

Caractéristiques Java [1 / 4]

▶ Simple

- Apprentissage facile
 - Faible nombre de mots-clés
 - Simplifications des fonctionnalités essentielles (Kit java & API)
- Développeurs opérationnels rapidement

▶ Familier

- Syntaxe proche de celle de C/C++

Caractéristiques Java [2 / 4]

► Orienté objet

- Java ne permet d'utiliser que des objets
- Java est un *langage objet* de la famille des langages de *classe* comme C++ ou SmallTalk
- Les grandes idées reprises sont : Encapsulation, attribut, méthode/message, visibilité, interface/implémentation, héritage simple, redéfinition de méthodes, polymorphisme

► Sûr

- Seul le bytecode est transmis, et «vérifié» par l'interpréteur
- Impossibilité d'accéder à des fonctions globales ou des ressources arbitraires du système

Caractéristiques Java [3 / 4]

► Fiable

- Gestion automatique de la mémoire (*ramasse-miette*)
- Philosophie de gestion des exceptions
- Sources d'erreurs limitées
 - Typage fort,
 - Pas d'héritage multiple,
 - Pas de manipulations de pointeurs.
- Vérifications faites par le compilateur facilitant une plus grande rigueur du code

Caractéristiques Java [4/4]

Java est indépendant de l'architecture

- ▶ Le format généré par le compilateur est indépendant de toute architecture.
- ▶ Rôle d'une machine virtuelle Java:

« Write once run everywhere »

Java est multi-tâches

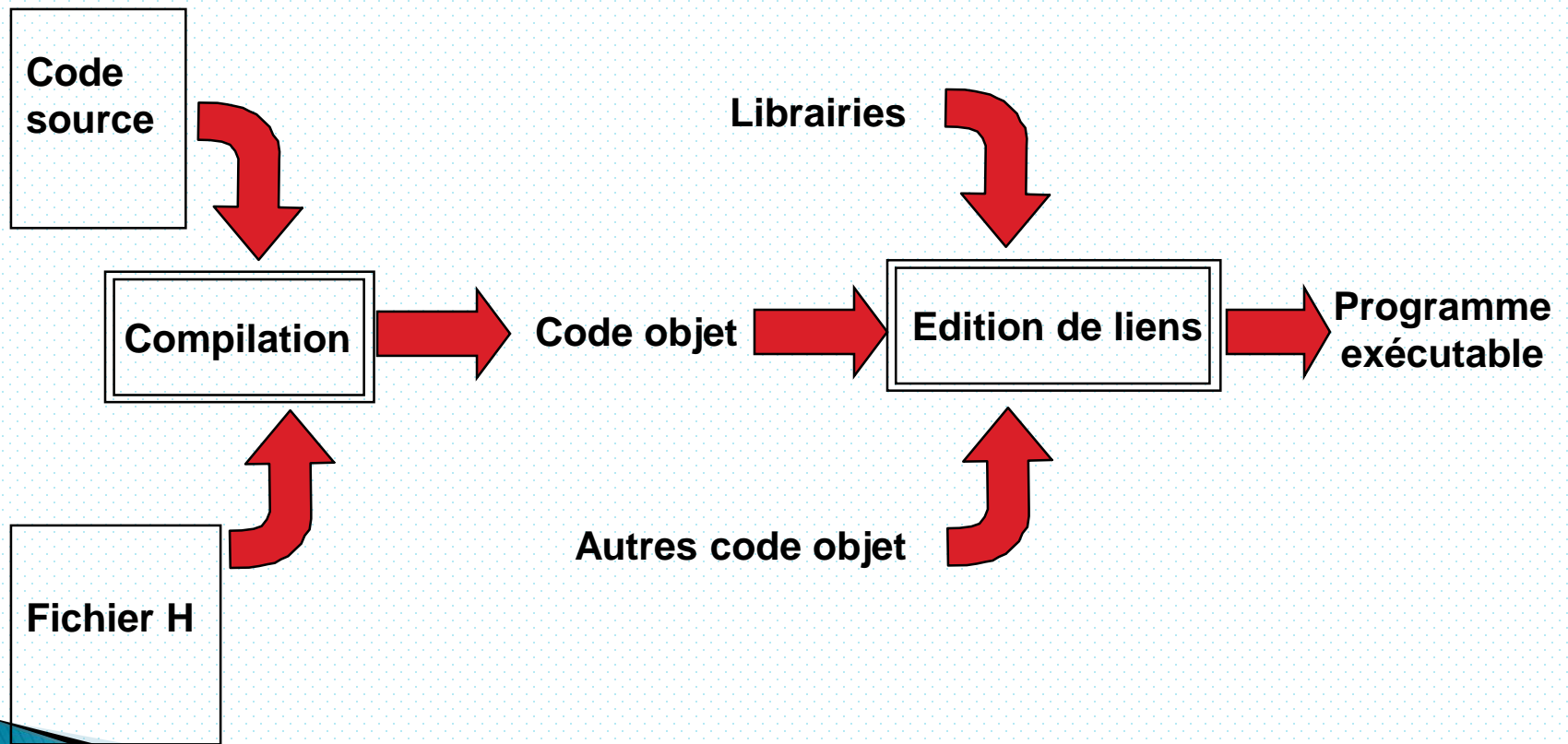
- ▶ Exécution de plusieurs processus effectuant chacun une tâche différente
- ▶ Mécanismes de synchronisation (Ressources)

Java, un langage indépendant

- ▶ Java est un langage interprété
 - La compilation d'un programme Java crée du pseudocode portable: le *bytecode*
 - Sur n'importe quelle plate-forme, une machine virtuelle Java peut interpréter le *bytecode* afin qu'il soit exécuté
 - Think to « **JVM** ».

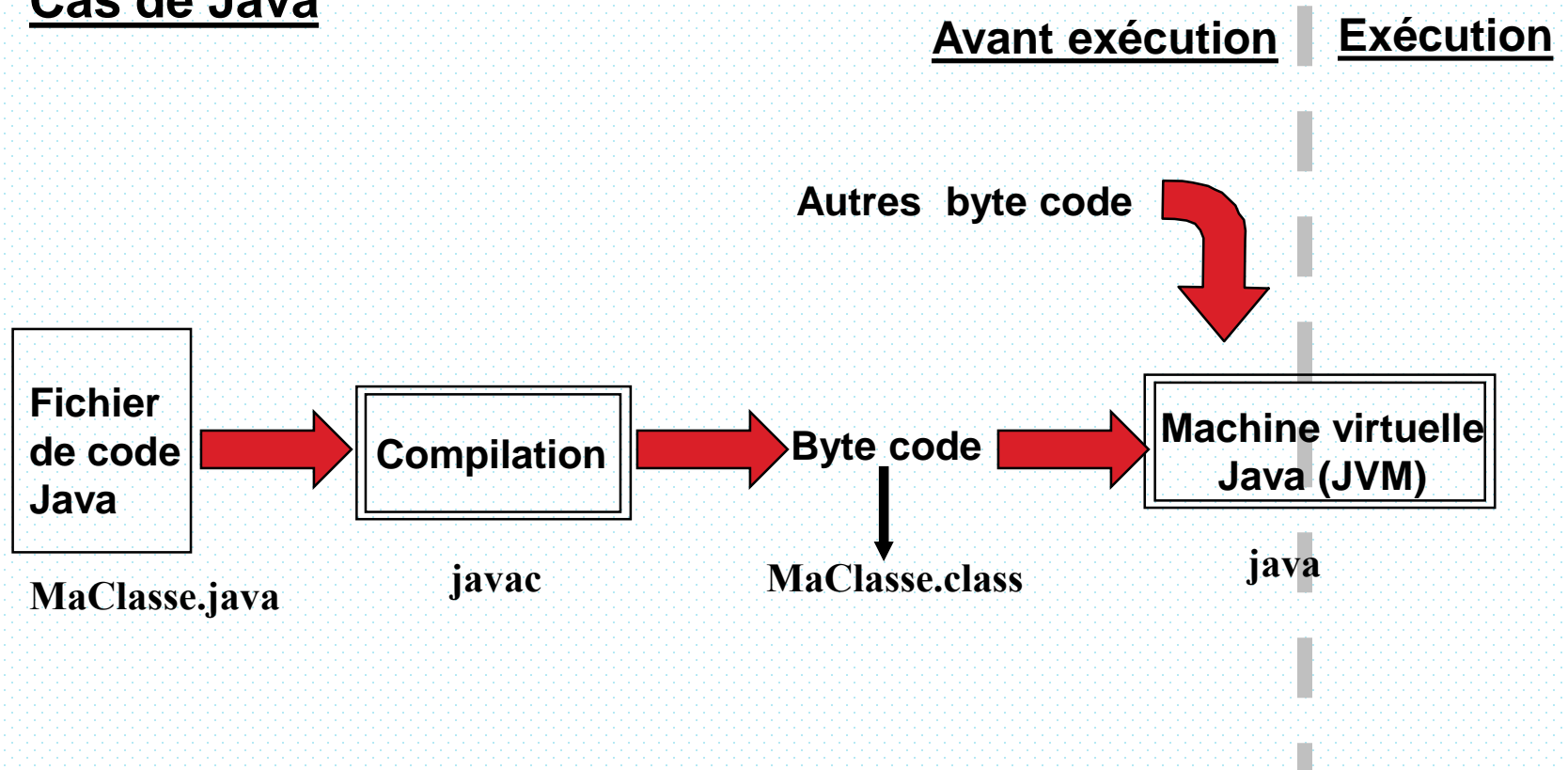
Langage compilé

Execution d'un langage compilé comme C++



Langage interprété

Cas de Java



L'API de Java [1 / 6]

- ▶ Java fournit de nombreuses librairies de classes remplissant des fonctionnalités très diverses : **C'est l'API Java**
- ▶ Ces classes sont regroupées, par catégories, en paquetages (ou "packages").

L'API de Java [2 / 6]

- ▶ Les principaux paquetages
 - **java.util** : structures de données classiques
 - **java.io** : entrées / sorties
 - **java.lang** : chaînes de caractères, interaction avec l'OS, threads
 - **java.applet** : les applets sur le web
 - **java.awt** : interfaces graphiques, images et dessins
 - **javax.swing** : package proposant des composants « légers » pour la création d'interfaces graphiques
 - **java.net** : sockets, URL
 - **java.sql** : fournit le package JDBC

L'API de Java [3 / 6]

- ▶ La documentation de Java est standard, que ce soit pour les classes de l'API ou pour les classes utilisateur
 - possibilité de génération automatique avec l'outil Javadoc.
- ▶ Elle est au format HTML.
 - Profiter des liens hypertexte pour Naviguer dans la documentation

L'API de Java [4 / 6]

- ▶ Pour chaque classe, il y a une page HTML contenant :
 - la hiérarchie d'héritage de la classe,
 - une description de la classe et son but général,
 - la liste des attributs de la classe (locaux et hérités),
 - la liste des constructeurs de la classe (locaux et hérités),
 - la liste des méthodes de la classe (locaux et hérités),
 - puis, chacune de ces trois dernières listes, avec la description détaillée de chaque élément.

L'API de Java [5 / 6]

► Où trouver les informations sur les classes de l'API

- Sous le répertoire `jdk1.x/docs/api` dans le JDK
 - les documentations de l'API se téléchargent et s'installent (en général) dans le répertoire dans lequel on installe java.

Par exemple si vous avez installé Java dans le répertoire ***D:/dev/apps/jdk1.8/***, vous décompresser le fichier zip contenant les documentations dans ce répertoire.

Les docs de l'API se trouveront alors sous :
D:/dev/apps/jdk1.8/docs/api/index.html

- Sur le site de Sun, on peut la retrouver à <https://docs.oracle.com/javase/7/docs/api/>

L'API de Java (6)

jdk.accessibility (Java SE 10) x

← → ↻ | Sécurisé | <https://docs.oracle.com/javase/10/docs/api/jdk.accessibility-summary.html>

ERP hto-bigdata 192.168.1.44 Test d'envoi et de réce JIRA - Installation de CRISCO QNAP BPM ERP-PROD LWS LWS Webmail ERP-REC ERP-F

OVERVIEW **MODULE** PACKAGE CLASS USE TREE DEPRECATED INDEX HELP Java SE 10 & JDK 10


PREV MODULE NEXT MODULE FRAMES NO FRAMES ALL CLASSES

MODULE: DESCRIPTION | MODULES | PACKAGES | SERVICES

Module jdk.accessibility

Defines JDK utility classes used by implementors of Assistive Technologies.

Module Graph:



```

graph TD
    jdk.accessibility --> java.desktop
    jdk.accessibility --> java.awt
    java.desktop --> java.awt.datatransfer
    java.awt --> java.awt.datatransfer
  
```

Since:
9

Packages

Exports

Package	Description
com.sun.java.accessibility.util	Provides a collection of interfaces and classes that compose the Java Accessibility Utilities.

Indirect Exports

From	Packages
java.datatransfer	java.awt.datatransfer
java.desktop	java.applet java.awt java.awt.color java.awt.desktop java.awt.dnd java.awt.event java.awt.font java.awt.geom java.awt.im java.awt.im.spi java.awt.image java.awt.image.renderable java.awt.print java.beans java.beans.beancontext javax.accessibility javax.imageio javax.imageio.event javax.imageio.metadata javax.imageio.plugins.bmp javax.imageio.plugins.jpeg javax.imageio.plugins.tiff javax.imageio.spi javax.imageio.stream javax.print javax.print.attribute javax.print.attribute.standard javax.print.event javax.sound.midi javax.sound.midi.spi javax.sound.sampled javax.sound.sampled.spi javax.swing javax.swing.border javax.swing.colorchooser javax.swing.event javax.swing.filechooser javax.swing.plaf javax.swing.plaf.basic javax.swing.plaf.metal javax.swing.plaf.multi

Outil de développement : Le JDK

- ▶ Environnement de développement fourni par Sun
- ▶ JDK signifie Java Development Kit (Kit de développement Java).
- ▶ Il contient :
 - les classes de base de l'API java (plusieurs centaines),
 - La documentation au format HTML
 - Le compilateur : Commande *javac*
 - La JVM (machine virtuelle) : commande *java*
 - Le générateur de documentation : commande *javadoc*

Java, un langage novateur ?

- ▶ Java n'est pas un langage novateur : il a puisé ses concepts dans d'autres langages existants et sa syntaxe s'inspire de celle du C++.
- ▶ Cette philosophie permet à Java
 - De ne pas dérouter ses utilisateurs en faisant *"presque comme ... mais pas tout à fait"*
 - D'utiliser des idées, concepts et techniques qui ont fait leurs preuves et que les programmeurs savent utiliser
- ▶ En fait, Java a su faire une synthèse efficace de bonnes idées issues de sources d'inspiration variées
 - Smalltalk, C++, Ada, etc.

Syntaxe du langage Java

Les commentaires

- ▶ `/* commentaire sur une ou plusieurs lignes */`
 - Identiques à ceux existant dans le langage C
- ▶ `// commentaire de fin de ligne`
 - Identiques à ceux existant en C++
- ▶ `/** commentaire d'explication */`
 - Les commentaires d'explication se placent généralement juste avant une déclaration (d'attribut ou de méthode)
 - Ils sont récupérés par l'utilitaire javadoc et inclus dans la documentation ainsi générée.

Instructions, blocs et blancs

- ▶ Les instructions Java se terminent par un ;
- ▶ Les blocs sont délimités par :

{ pour le début de bloc

} pour la fin du bloc

Un bloc permet de définir un regroupement d'instructions. La définition d'une classe ou d'une méthode se fait dans un bloc.

- ▶ Les espaces, tabulations, sauts de ligne sont autorisés. Cela permet de présenter un code plus lisible.


Point d'entrée d'un programme Java

- ▶ Pour pouvoir faire un programme exécutable il faut toujours une classe qui contienne une méthode particulière, la méthode « main »
 - c'est le point d'entrée dans le programme : le microprocesseur sait qu'il va commencer à exécuter les instructions à partir de cet endroit

```
public static void main(String arg[ ]){  
    ...  
    ...  
}
```


Exemple (1)

Fichier HelloWorld.java



```
public class HelloWorld{
//Accolade débutant la classe HelloWorld
    public static void main(String args[])
    { //Accolade débutant la méthode main

        /* Pour l'instant juste une instruction */
        System.out.println("HelloWord");
    } //Accolade fermant la méthode main
} //Accolade fermant la classe Bonjour
```



Compilation et exécution (1)

Fichier `HelloWord.java`

Le nom du fichier est nécessairement celui de la classe avec l'extension `.java` en plus. Java est sensible à la casse des lettres.

Compilation en bytecode java dans une console DOS:

`javac HelloWord.java`

Génère un fichier

`HelloWord.class`

Exécution du programme (toujours depuis la console DOS) sur la JVM :

`java HelloWord`

Affichage de « HelloWord » dans la console

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

Identificateurs [1 / 2]

- ▶ On a besoin de nommer les classes, les variables, les constantes, etc. ; on parle d'identificateur.
- ▶ Les identificateurs commencent par une lettre, _ ou \$
*Attention : **Java est sensible à la casse***
- ▶ Conventions sur les identificateurs :
 - Si l'identificateur est un mot composé, la première lettre de chaque mot est en majuscule excepté le premier.
 - exemple : uneVariableEntiere
 - La première lettre est majuscule pour les classes et les interfaces
 - exemples : NotreInterface, NotreClasseImpl

Identificateurs [2 / 2]

- ▶ Conventions sur les identificateurs :
 - La première lettre est minuscule pour les méthodes, les attributs et les variables
 - exemples : setNom, i, uneBelleVoiture
 - Les constantes sont entièrement en majuscules
 - exemple : LONGUEUR_MAX

Les mots réservés à Java

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

Les types primitifs [1 / 12]

- ▶ En Java, tout est objet sauf les types de base.
- ▶ Il y a huit types de base :
 - Un type booléen pour représenter les variables ne pouvant prendre que 2 valeurs (vrai et faux, 0 ou 1, etc.) : **boolean** avec les valeurs associées **true** et **false**
 - Un type pour représenter les caractères : **char**
 - Quatre types pour représenter les entiers de divers taille : **byte**, **short**, **int** et **long**
 - Deux types pour représenter les réelles : **float** et **double**
- ▶ La taille nécessaire au stockage de ces types est indépendante de la machine.
 - Avantage : Portabilité
 - Inconvénient : "conversions" coûteuses

Les types primitifs [2 / 12]

→ Les entiers

► Les entiers (avec signe)

- **byte** : codé sur 8 bits, peuvent représenter des entiers allant de -2^7 à $2^7 - 1$ (-128 à $+127$)
- **short** : codé sur 16 bits, peuvent représenter des entiers allant de -2^{15} à $2^{15} - 1$
- **int** : codé sur 32 bits, peuvent représenter des entiers allant de -2^{31} à $2^{31} - 1$
- **long** : codé sur 64 bits, peuvent représenter des entiers allant de -2^{63} à $2^{63} - 1$

Les types primitifs [3 / 12]

→ Les entiers

▶ Notation

- 2 entier normal en base décimal
- 2L entier au format long en base décimal
- **0**10 entier en valeur octale (base 8)
- **0x**F entier en valeur hexadécimale (base 16)

▶ Opérations sur les entiers

- Opérateurs arithmétiques $+$, $-$, $*$, $/$
- $/$: Division entière si les 2 arguments sont des entiers
- $\%$ (Modulo): Reste de la division entière
 - exemples :
 - $15 / 4$ donne 3
 - $15 \% 2$ donne 1

Les types primitifs [4/12]

→ Les entiers

► Opérations sur les entiers (suite)

◦ les opérateurs d'incrémentation ++ et de décrémentation --

- Ajoute ou retranche 1 à une variable

```
int n = 12;
```

```
n ++; //Maintenant n vaut 13
```

- `n++`; « équivalent à » `n = n+1`;
- `n--`; « équivalent à » `n = n-1`;
- `8++`; est une instruction illégale
- Peut s'utiliser de manière suffixée : `++n`. La différence avec la version préfixée se voit quand on les utilisent dans les expressions.

En version préfixée la (dé/inc)rémentation s'effectue en premier

```
int m=7; int n=7;  
int a=2 * ++m; //a vaut 16, m vaut 8  
int b=2 * n++; //b vaut 14, n vaut 8
```

Les types de bases (5) : les réels

► Les réels

- float : codé sur 32 bits, peuvent représenter des nombres allant de -10^{35} à $+10^{35}$
- double : codé sur 64 bits, peuvent représenter des nombres allant de -10^{400} à $+10^{400}$

► Notation

- 4.55 ou 4.55**D** réel double précision
- 4.55**f** réel simple précision

Les types de bases (6) : les réels

► Les opérateurs

- opérateurs classiques $+$, $-$, $*$, $/$
- attention pour la division :
 - $15 / 4$ donne 3 *division entière*
 - $15 \% 2$ donne 1
 - $11.0 / 4$ donne 2.75
(si l'un des termes de la division est un réel, la division retournera un réel).
- puissance : utilisation de la méthode `pow` de la classe `Math`.
 - `double y = Math.pow(x, a)` équivalent à x^a ,
 x et a étant de type `double`

Les types de bases (7) : les booléens

▶ Les booléens

- boolean
contient soit vrai (**true**) soit faux (**false**)

▶ Les opérateurs logiques de comparaisons

- Egalité : opérateur **==**
- Différence : opérateur **!=**
- supérieur et inférieur strictement à :
opérateurs **>** et **<**
- supérieur et inférieur ou égal :
opérateurs **>=** et **<=**

Les types de bases (8) : Les booléens

► Notation

boolean x;

x= true;

x= false;

x= (5==5); // l'expression (5==5) est évaluée et la valeur est affectée à x qui vaut alors vrai

x= (5!=4); // x vaut vrai, ici on obtient vrai si 5 est différent de 4

x= (5>5); // x vaut faux, 5 n'est pas supérieur strictement à 5

x= (5<=5); // x vaut vrai, 5 est bien inférieur ou égal à 5

Les types de bases (9) : Les booléens

- ▶ Les autres opérateurs logiques
 - et logique : **&&**
 - ou logique : **||**
 - non logique : **!**
 - Exemples : si a et b sont 2 variables booléennes

```
boolean a,b, c;
```

```
a= true;
```

```
b= false;
```

```
c= (a && b); // c vaut false
```

```
c= (a || b); // c vaut true
```

```
c= !(a && b); // c vaut true
```

```
c=!a; // c vaut false
```

Les types de bases (10) : les caractères

► Les caractères

- **char** : contient une seule lettre
- le type char désigne des caractères en représentation Unicode
 - Codage sur 2 octets contrairement à ASCII/ANSI codé sur 1 octet. Le codage ASCII/ANSI est un sous-ensemble d'Unicode
 - Notation hexadécimale des caractères Unicode de ' \u0000 ' à ' \uFFFF '.
 - Plus d'information sur Unicode à : www.unicode.org

Les types de bases (1 1) : les caractères

► Notation

char a,b,c; // a,b et c sont des variables du type char

a='a'; // a contient la lettre 'a'

b= '\u0022' //b contient le caractère *guillemet* : "

c=97; // x contient le caractère de rang 97 : 'a'

Les types de bases (12)

exemple et remarque

```
int x = 0, y = 0;  
float z = 3.1415F;  
double w = 3.1415;  
long t = 99L;  
boolean test = true;  
char c = 'a';
```

- ▶ Remarque importante :
 - Java exige que toutes les variables soient définies et initialisées. Le compilateur sait déterminer si une variable est susceptible d'être utilisée avant initialisation et produit une erreur de compilation.

Les structures de contrôles (1)

- ▶ Les structures de contrôle classiques existent en Java :
 - **if, else**
 - **switch, case, default, break**
 - **for**
 - **while**
 - **do, while**

Les structures de contrôles (2) : if / else

► Instructions conditionnelles

- Effectuer une ou plusieurs instructions seulement si une certaine condition est vraie

if (*condition*) *instruction*;

et plus généralement : **if** (*condition*)
 { *bloc d'instructions* }

condition doit être un booléen ou renvoyer une valeur booléenne

- Effectuer une ou plusieurs instructions si une certaine condition est vérifiée sinon effectuer d'autres instructions

if (*condition*) *instruction1*; **else** *instruction2*;

et plus généralement **if** (*condition*) { *1^{er} bloc d'instructions* } **else** { *2^{ème} bloc d'instruction* }

Les structures de contrôles (3) : if /

```
Max.java ✖
1 package fr.htc.iffelse;
2
3 import java.util.Scanner;
4
5 public class Max {
6     public static void main(String args[]) {
7         Scanner scanner = new Scanner(System.in);
8
9         System.out.print("Entrer un entier : ");
10        int nb1 = scanner.nextInt();
11
12        System.out.print("Entrer un autre entier : ");
13        int nb2 = scanner.nextInt();
14
15        if (nb1 > nb2) {
16            System.out.println("l'entier le plus grand est " + nb1);
17        } else {
18            System.out.println("l'entier le plus grand est " + nb2);
19        }
20
21        scanner.close();
22    }
23 }
```


Les structures de contrôles (4) : while

► Boucles indéterminées

- On veut répéter une ou plusieurs instructions un nombre indéterminés de fois : on répète l'instruction ou le bloc d'instruction tant que une certaine condition reste vraie
- nous avons en Java une première boucle while (tant que)
 - **while** (*condition*) {*bloc d'instructions*}
 - les instructions dans le bloc sont répétées tant que la condition reste vraie.
 - On ne rentre jamais dans la boucle si la condition est fausse dès le départ

Les structures de contrôles (5) : while

► Boucles indéterminées

- un autre type de boucle avec le while:
 - **do** {*bloc d'instructions*} **while** (*condition*)
 - les instructions dans le bloc sont répétées tant que la condition reste vraie.
 - On rentre toujours au moins une fois dans la boucle : la condition est testée en fin de boucle.

Les structures de contrôles (6)

: while

Factoriel.java

```
1 package fr.htc.iffelse;
2
3 import java.util.Scanner;
4
5 public class Factoriel {
6
7     public static void main(String args[]) {
8         Scanner scanner = new Scanner(System.in);
9         long n, result, i;
10
11         System.out.print("Entrer une valeur pour n: ");
12         n = scanner.nextInt();
13
14         result = 1;
15         i = n;
16         while (i > 1) {
17             result = result * i;
18             i--;
19         }
20         System.out.println("la factorielle de " + n + " vaut " + result);
21         scanner.close();
22     }
23 }
```

Les structures de contrôles (7) : for

► Boucles déterminées

- On veut répéter une ou plusieurs instructions un nombre déterminés de fois : on répète l'instruction ou le bloc d'instructions pour un certain nombre de pas.

- La boucle for

```
for (int i = 1; i <= 10; i++)
```

```
    System.out.println(i); //affichage des nombres de 1 à 10
```

- une boucle for est en fait équivalente à une boucle while

```
for (instruction1; expression1; expression2) {bloc}
```

... est équivalent à ...

```
instruction 1; while (expression1) {bloc; expression2}}
```

Les structures de contrôles (8) : for

Facto2.java

```
import java.io.*;

public class Facto2
{
    public static void main(String args[])
    {
        int n, result,i;
        n = Integer.parseInt(System.console().readLine("Entrer une valeur pour n:"));
        result = 1;
        for(i =n; i > 1; i--)
        {
            result = result * i;
        }
        System.out.println("la factorielle de "+n+" vaut "+result);
    }
}
```

Les structures de contrôles (9) : switch

► Sélection multiples

- l'utilisation de if / else peut s'avérer lourde quand on doit traiter plusieurs sélections et de multiples alternatives
- pour cela existe en Java le **switch** / **case** assez identique à celui de C/C++
- La valeur sur laquelle on teste doit être un char ou un entier (à l'exclusion d'un **long**) ou une chaîne de caractère ou encore une énumération.
- L'exécution des instructions correspondant à une alternative commence au niveau du **case** correspondant et se termine à la rencontre d'une instruction **break** ou arrivée à la fin du **switch**

Les structures de contrôles (10) : switch

Alternative.java

```
import java.io.*;

public class Alternative
{
    public static void main(String args[])
    {
        int nb = Integer.parseInt(System.console().readLine("Entrer une valeur pour n:"));
        switch(nb)
        {
            case 1:
                System.out.println("Un"); break;
            case 2:
                System.out.println("Deux"); break;
            default:
                System.out.println("Autre nombre"); break;
        }
    }
}
```

Variable contenant la valeur que l'on veut tester.

Première alternative : on affiche *Un* et on sort du bloc du switch au break;

Deuxième alternative : on affiche *Deux* et on sort du bloc du switch au break;

Alternative par défaut: on réalise une action par défaut.

Les tableaux (1)

- ▶ Les tableaux permettent de stocker plusieurs valeurs de même type dans une variable.
 - Les valeurs contenues dans la variable sont repérées par un indice
 - En langage java, les tableaux sont des objets
- ▶ Déclaration
 - `int tab [];`
`String noms[];`
- ▶ Création d'un tableau
 - `tab = new int [20]; // tableau de 20 int`
 - `noms= new String [100]; // tableau de 100 chaine`

Les tableaux (2)

- ▶ Le nombre d'éléments du tableau est mémorisé. Java peut ainsi détecter à l'exécution le dépassement d'indice et générer une exception. Mot clé **length**
 - Il est récupérable par **nomTableau.length**
`int taille = tab.length; //taille vaut 20`
- ▶ Comme en C/C++, les indices d'un tableau commencent à '0'. Donc un tableau de taille 100 aura ses indices qui iront de 0 à 99.

Les tableaux (3)

- ▶ Initialisation

```
tab[0]=1;
```

```
tab[1]=2; //etc.
```

```
noms[0] = new String( "Boule");
```

```
noms[1] = new String( "Bill");//etc
```

- ▶ Création et initialisation simultanées

```
String noms [ ] = {"Boule","Bill"};
```

```
Point pts[ ] = { new Point (0, 0), new Point (10, -1)};
```

Les tableaux (4)

Tab1.java

```
public class Tab1
{
    public static void main (String args[])
    {
        int tab[ ] ;
        tab = new int[4];
        tab[0]=5;
        tab[1]=3;
        tab[2]=7;
        tab[3]=tab[0]+tab[1];
    }
}
```

Pour déclarer une variable tableau on indique le *type* des éléments du tableau et le *nom* de la *variable tableau* suivi de `[]`

on utilise `new <type> [taille];` pour initialiser le tableau

On peut ensuite affecter des valeurs au différentes cases du tableau :
`<nom_tableau>[indice]`

Les indices vont toujours de 0 à (taille-1)

Les tableaux (5)

Tab1.java

```
public class Tab1
{
    public static void main (String args[])
    {
        int tab[ ] ;
        tab = new int[4];
        tab[0]=5;
        tab[1]=3;
        tab[2]=7;
        tab[3]=tab[0]+tab[1];
    }
}
```

Mémoire

0x258

0
0
0
0

Les tableaux (6)

Tab1.java

```
public class Tab1
{
    public static void main (String args[])
    {
        int tab[ ] ;
        tab = new int[4];
        tab[0]=5;
        tab[1]=3;
        tab[2]=7;
        tab[3]=tab[0]+tab[1];
    }
}
```

Mémoire

0x258

5

3

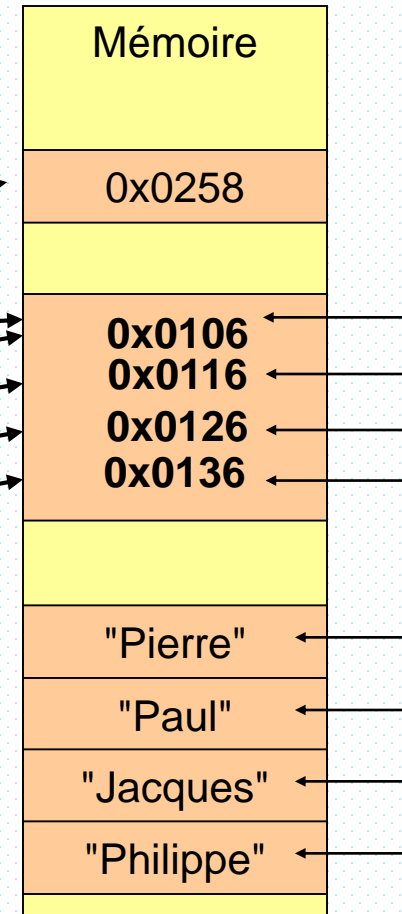
7

8

Les tableaux (7)

Tab2.java

```
public class Tab2
{
    public static void main (String args[])
    {
        String tab[ ];
        tab = new String[4];
        tab[0]=new String("Pierre");
        tab[1]=new String("Paul");
        tab[2]=new String(" Jacques");
        tab[3]=new String("Philippe");
    }
}
```



La classe String (1)

- ▶ Attention ce n'est pas un type de base. Il s'agit d'une classe défini dans l'API Java (Dans le package java.lang)

String s="aaa"; // s contient la chaîne "aaa"

String s=**new String**("aaa"); // identique à la ligne précédente

- ▶ La concaténation

- l'opérateur **+** entre 2 String les concatène :

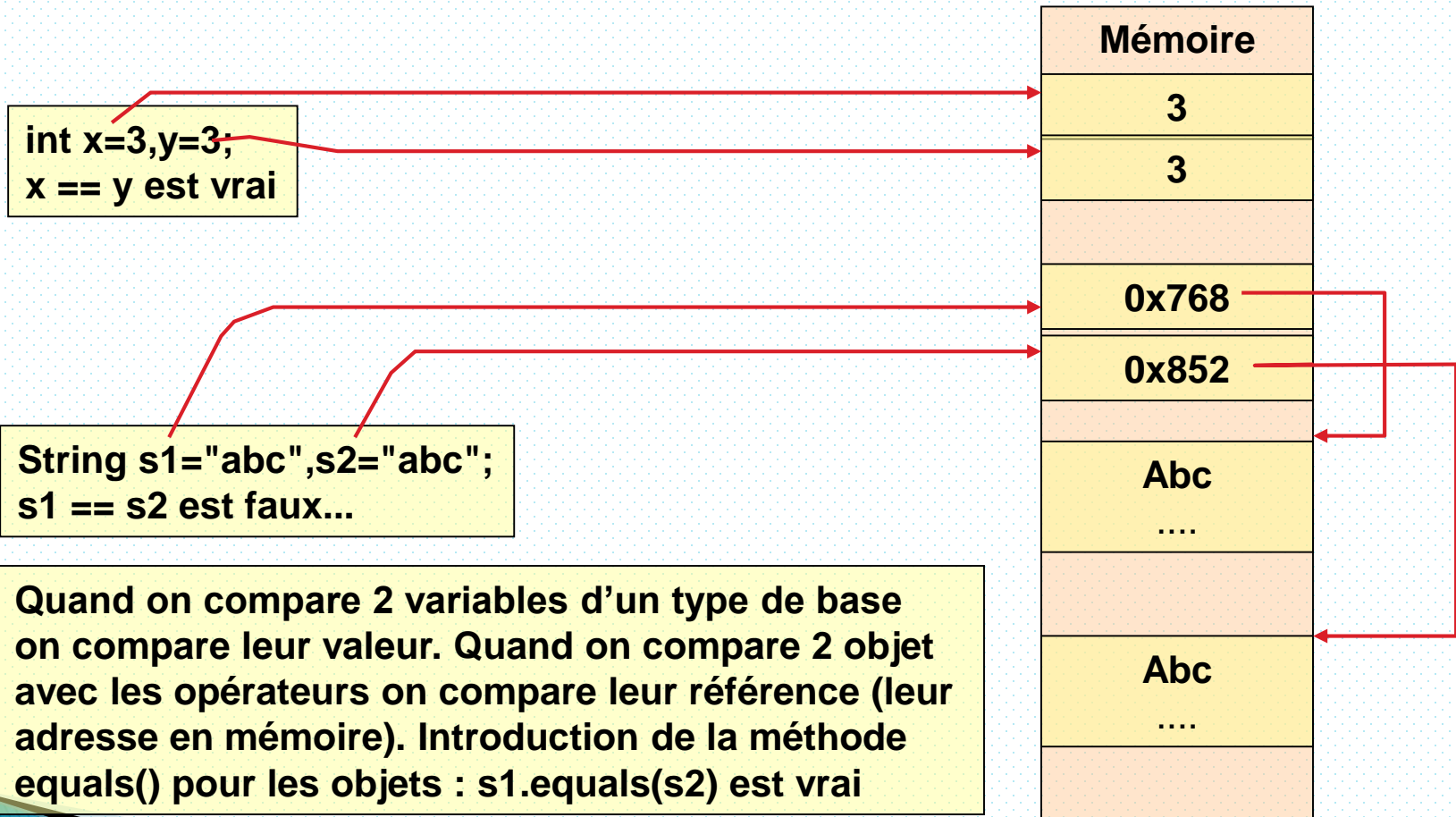
String str1 = "Bonjour ! ";

String str2 = null;

str2 = "Comment vas-tu ?";

String str3 = str1 + str2; /* Concaténation de chaînes : str3 contient " Bonjour ! Comment vas-tu ?" */

Différences entre objets et types de base



La classe String (2)

- ▶ Longueur d'un objet String :
 - méthode `int length()` : renvoie la longueur de la chaîne
`String str1 = "bonjour";`
`int n = str1.length(); // n vaut 7`
- ▶ Sous-chaînes
 - méthode `String substring(int debut, int fin)`
 - extraction de la sous-chaîne depuis la position `debut` jusqu'à la position `fin` non-comprise.
`String str2 = str1.substring(0,3); // str2 contient la valeur "bon"`
 - le premier caractère d'une chaîne occupe la position 0
 - le deuxième paramètre de `substring` indique la position du premier caractère que l'on ne souhaite pas copier

La classe String (3)

- ▶ Récupération d'un caractère dans une chaîne
 - méthode `char charAt(int pos)` : renvoie le caractère situé à la position `pos` dans la chaîne de caractère à laquelle on envoie le message

```
String str1 = "bonjour";
char unJ = str1.charAt(3); // unJ contient le caractère 'j'
```
- ▶ Modification des objets String
 - Les String sont inaltérables en Java : on ne peut modifier individuellement les caractères d'une chaîne.
 - Par contre il est possible de modifier le contenu de la variable contenant la chaîne (la variable ne référence plus la même chaîne).

```
str1 = str1.substring(0,3) + "soir"; /* str1 contient maintenant la
chaîne "bonsoir" */
```

La classe String (4)

- ▶ Les chaînes de caractères sont des objets :
 - pour tester si 2 chaînes sont égales il faut utiliser la méthode `boolean equals(String str)` et non `==`
 - pour tester si 2 chaînes sont égales à la casse près il faut utiliser la méthode `boolean equalsIgnoreCase(String str)`

```
String str1 = "BonJour";
```

```
String str2 = "bonjour"; String str3 = "bonjour";
```

```
boolean a, b, c, d;
```

```
a = str1.equals("BonJour"); //a contient la valeur true
```

```
b = (str2 == str3); //b contient la valeur false
```

```
c = str1.equalsIgnoreCase(str2); //c contient la valeur true
```

```
d = "bonjour".equals(str2); //d contient la valeur true
```

La classe String (5)

- ▶ Quelques autres méthodes utiles
 - `boolean startsWith(String str)` : pour tester si une chaîne de caractère commence par la chaîne de caractère `str`
 - `boolean endsWith(String str)` : pour tester si une chaîne de caractère se termine par la chaîne de caractère `str`
- ```
String str1 = "bonjour ";
boolean a = str1.startsWith("bon");//a vaut true
boolean b = str1.endsWith("jour");//b vaut true
```