PROGRAMMATION ORIENTÉE OBJET

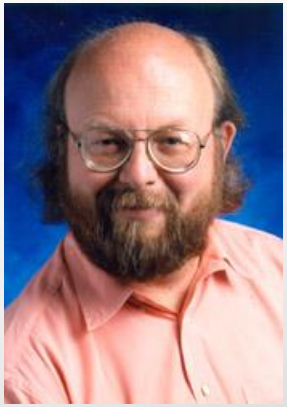
LANGAGE JAVA

SONIA KEFI

2 ÈME LI

ANNÉE UNIVERSITAIRE : 2024-2025

HISTORIQUE DE JAVA (1)



- Java a été développé à partir de décembre 1990 par une équipe de Sun Microsystems dirigée par James Gosling.
- Au départ, il s'agissait de développer un langage de programmation pour permettre le dialogue entre de futurs ustensiles domestiques.

- Or, les langages existants tels que C++ ne sont pas à la hauteur : recompilation dès qu'une nouvelle puce arrive, complexité de programmation pour l'écriture de logiciels fiables...

HISTORIQUE DE JAVA (2)

- 1990 : Ecriture d'un nouveau langage plus adapté à la réalisation de logiciels embarqués, appelé OAK
  - Petit, fiable et indépendant de l'architecture
  - Destiné à la télévision interactive
  - Non rentable sous sa forme initiale
- 1993 : le WEB « décolle », Sun redirige ce langage vers Internet : les qualités de portabilité et de compacité du langage OAK en ont fait un candidat parfait à une utilisation sur le réseau. Cette réadaptation prit près de 2 ans.
- 1995 : Sun rebaptisa OAK en Java (nom de la machine à café autour de laquelle se réunissait James Gosling et ses collaborateurs)

LES DIFFÉRENTES VERSIONS DE JAVA

- De nombreuses versions de Java depuis 1995
  - Java 1.0 en 1995
  - Java 1.1 en 1996
  - Java 1.2 en 1999 (Java 2, version 1.2)
  - Java 1.3 en 2001 (Java 2, version 1.3)
  - Java 1.4 en 2002 (Java 2, version 1.4)
  - Java 1.5 en 2004
  - Java 1.6 en 2006
  - Java 1.7 en 2011
  - Java 1.8 en 2014
  - Java 9 en 2018
  - …
  - Java 14 en 17 Mars 2020
  - Java 15 en 15 Septembre 2020
  - Java 16 en 16 Mars 2021
  - Java 17 en 14 Septembre 2021
  - Java 18 en 22 Mars 2022
  - Java 19 en Septembre 2022
  - Java 24 en 2023
  - …..
- Évolution très rapide et succès du langage.
- Une certaine maturité atteinte avec Java 2.
- Mais des problèmes de compatibilité existaient :
  - entre les versions 1.1 et 1.2/1.3/1.4

HISTOIRE RÉCENTE

- En mai 2007, Sun publie l'ensemble des outils Java dans un « package » OpenJDK sous licence libre.
- La société Oracle a acquis en 2009 l'entreprise Sun Microsystems. On peut désormais voir apparaître le logo Oracle dans les documentations de l'api Java.
- Le 12 avril 2010, James Gosling, le créateur du langage de programmation Java démissionne d'Oracle pour des motifs qu'il ne souhaite pas divulguer. Il était devenu le directeur technologique de la division logicielle client pour Oracle.
- Août 2012 : il y a eu une faille de sécurité importante dans Java 7.

QUE PENSER DE JAVA?

- Il a su s'imposer dans de nombreux domaines.
- Un environnement gratuit et de nombreux outils disponibles.
- Une large communauté très active.
- Un bon langage.
- Un langage adapté aux besoins de développements actuels.

ORGANISATION DU COURS

- Nous verrons :
  - Caractéristiques de Java et son environnement de développement
  - Structures fondamentales
  - La programmation par objets en Java
    - **Héritage**
    - **Polymorphisme**
    - **Encapsulation**
  - Les exceptions, les entrées / sorties en Java
  - Les collections en Java
  - Les paquetages
- Nous essaierons d'aborder les thèmes suivants, si nous en avons le temps :
  - La programmation des interfaces graphiques en Java (AWT et Swing),
  - Les threads (appelés aussi processus légers)
  - L'accès aux bases de données

RÉFÉRENCES (1)

- **Bibliographie**
  - Au cœur de Java 2 : Volume I - Notions fondamentales. C. Hortsmann et G. Cornell. The Sun Microsystems Press. Java Series. CampusPress.
  - Au cœur de Java 2 : Volume II - Fonctions avancées. C. Hortsmann et G. Cornell. The Sun Microsystems Press. Java Series. CampusPress.
  - Passeport pour l'algorithmique objet. Jean-Pierre Fournier. Thomsom Publishing International.

RÉFÉRENCES (2)

- **Webographie**
  - Pour récupérer le kit de développement de Sun
    - http://java.sun.com/ (racine du site)
  - Outils de développement
    - Eclipse : http://www.eclipse.org
  - Des exemples de programmes commentés
    - http://www.technobuff.com/javatips/ (en anglais)
    - http://developer.java.sun.com/developer/JDCTechTips/ (en anglais)

DU PROBLÈME AU PROGRAMME

- Nécessité d'analyser le problème pour pouvoir le traduire en une solution informatique.
- Avant de construire un bâtiment, on fait un plan. Ce n'est pas différent en informatique, il s'agit de la conception de l'algorithme.
- Puis la mise en œuvre technique - le codage - dans un langage de programmation, dans notre cas Java.

ANALYSE DU PROBLÈME

- Se poser les bonnes questions :
  - Quelles sont les objets qui interviennent dans le problème? Quelles sont les données que le programme va manipuler?
  - Quelles sont les relations entre ces objets? Quelles sont les opérations qu'on va pouvoir effectuer sur ces objets?

CARACTÉRISTIQUES DU LANGAGE JAVA

CARACTÉRISTIQUES DU LANGAGE JAVA (1)

- **Simple**
  - Apprentissage facile:
    - faible nombre de mots-clés,
    - simplification des fonctionnalités essentielles.
  - Développeurs opérationnels rapidement.

- **Familier**
  - Syntaxe proche de celle de C/C++.

CARACTÉRISTIQUES DU LANGAGE JAVA (2)

- Orienté objet
  - Java manipule que des objets (hors les types de base).
  - Java est un langage objet de la famille des langages de classe comme C++ ou SmallTalk.
  - Les grandes idées reprises sont : encapsulation, dualité classe /instance, attribut, méthode / message, visibilité, dualité interface/implémentation, héritage simple, redéfinition de méthodes, polymorphisme.

CARACTÉRISTIQUES DU LANGAGE JAVA (3)

- **Fiable**
  - Gestion automatique de la mémoire ("garbage collector")
  - Gestion des exceptions
  - Sources d'erreurs limitées
    - typage fort,
    - pas d'héritage multiple,
    - pas de manipulations de pointeurs, etc.
  - Vérifications faites par le compilateur facilitant une plus grande rigueur du code

CARACTÉRISTIQUES DU LANGAGE JAVA (4)

- Java est indépendant de l'architecture :

- Le bytecode généré par le compilateur est indépendant de toute architecture. Toute application peut donc tourner sur une plate-forme implémentant une machine virtuelle Java :

« Ecrire une fois, exécuter partout »

- Java est multi-tâches :

- Exécution de plusieurs processus effectuant chacun une tâche différente.
- Mécanismes de synchronisation.
- Fonctionnement sur des machines multiprocesseurs.

JAVA, UN LANGAGE INDÉPENDANT? (1)

- Java est un langage interprété :
  - La compilation d'un programme Java crée du pseudo-code portable : le "byte-code".
  - Sur n'importe quelle plate-forme, une machine virtuelle Java peut interpréter le pseudo-code afin qu'il soit exécuté.
- Les machines virtuelles Java peuvent être :
  - des interpréteurs de byte-code indépendants (pour exécuter les programmes Java).
  - contenues au sein d'un navigateur (pour exécuter des applets Java).

JAVA, UN LANGAGE INDÉPENDANT? (2)

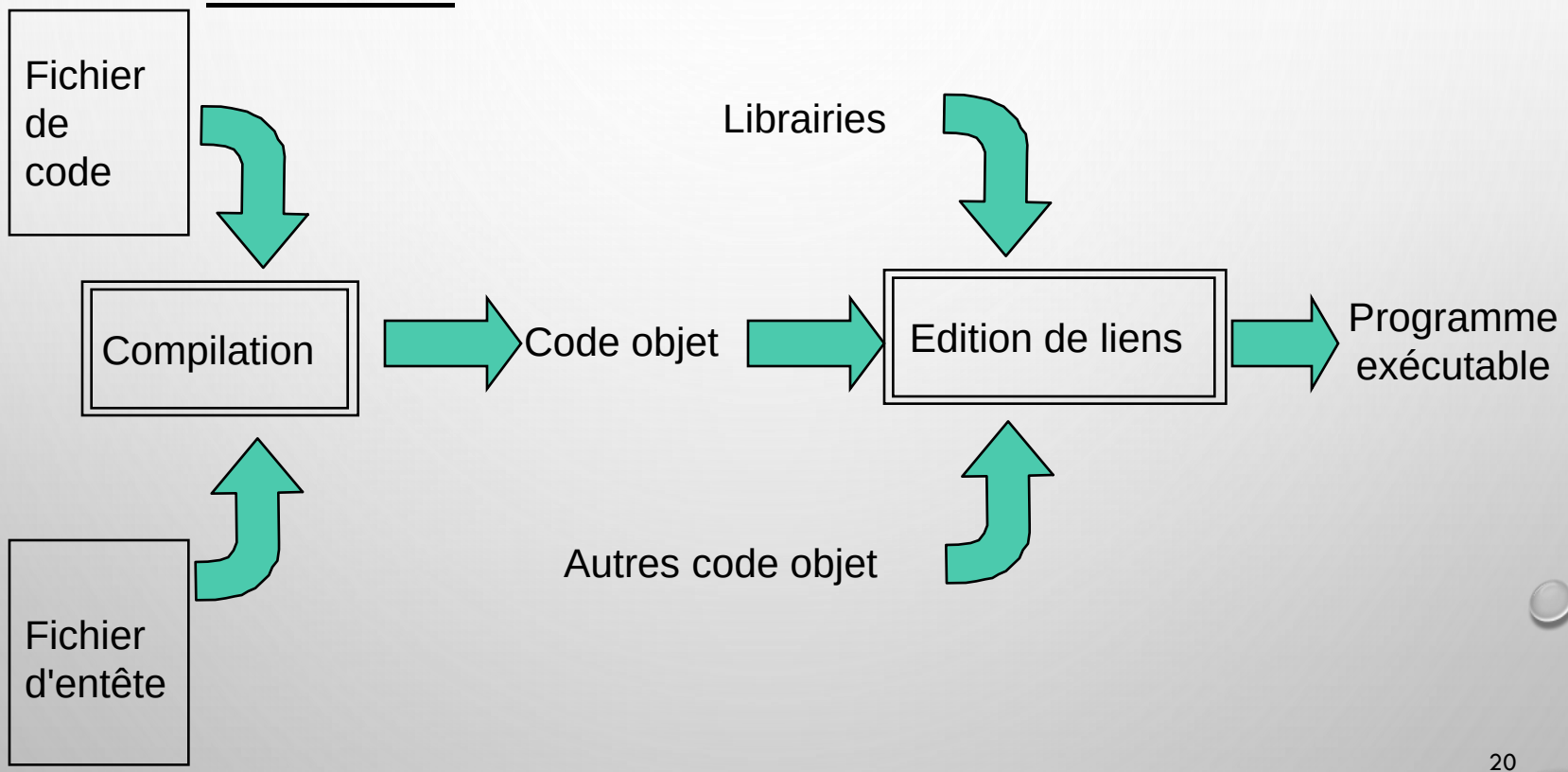
- Avantages :
  - **Portabilité**
    - Des machines virtuelles Java existent pour de nombreuses plates-formes dont : Linux, Windows, MacOS
  - **Développement plus rapide**
    - courte étape de compilation pour obtenir le byte-code,
    - pas d'édition de liens,
    - débogage plus aisé,
  - **Le byte-code** est plus compact que les exécutables :
    - pour voyager sur les réseaux.

JAVA, UN LANGAGE INDÉPENDANT? (3)

- **Inconvénients :**
  - Nécessite l'installation d'un interpréteur pour pouvoir exécuter un programme Java.
  - L'interprétation du code ralentit l'exécution.
  - Gestion gourmande de la mémoire.
  - Impossibilité d'opérations de « bas niveau » liées au matériel.

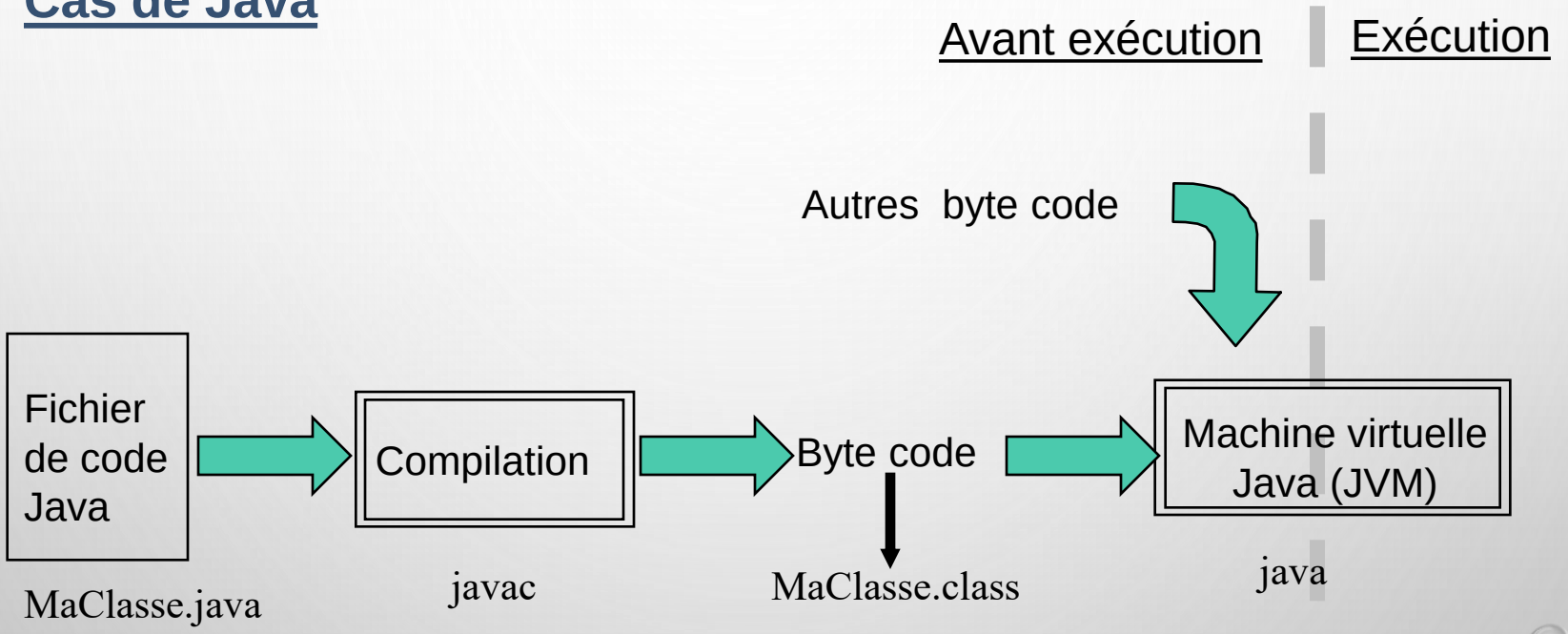
LANGAGE COMPILÉ

Etapes qui ont lieu avant l'exécution pour un langage compilé comme C++



LANGAGE INTERPRÉTÉ

Cas de Java



L'API DE JAVA (1)

- Java fournit de nombreuses librairies de classes remplissant des fonctionnalités très diverses : c'est l'API Java.
  - **API** (Application and Programming Interface /Interface pour la programmation d'applications) : Ensemble de bibliothèques permettant une programmation plus aisée car les fonctions deviennent indépendantes du matériel.
- Ces classes sont regroupées, par catégories, en **paquetages** (ou "packages").

L'API DE JAVA (2)

- **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 récent proposant des composants « légers » pour la création d'interfaces graphiques
  - java.net : sockets, URL
  - java.rmi : Remote Method Invocation
  - java.sql : fournit le package JDBC

L'API DE JAVA (3)

- 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 (4)

- 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 installer Java dans le répertoire D:/Apps/jdk1.4/, vous décompresser le fichier zip contenant les documentations dans ce répertoire.
    - Les docs de l'API se trouveront alors sous : D:/Apps/jdk1.4/docs/api/index.html
  - Sur le site de Sun, on peut la retrouver à http://java.sun.com/docs/index.html

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 : javac
  - la JVM (machine virtuelle) : java
  - le visualiseur d'applets : appletviewer
  - le générateur de documentation : javadoc
  - 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[ ])
{
…/…
}
```

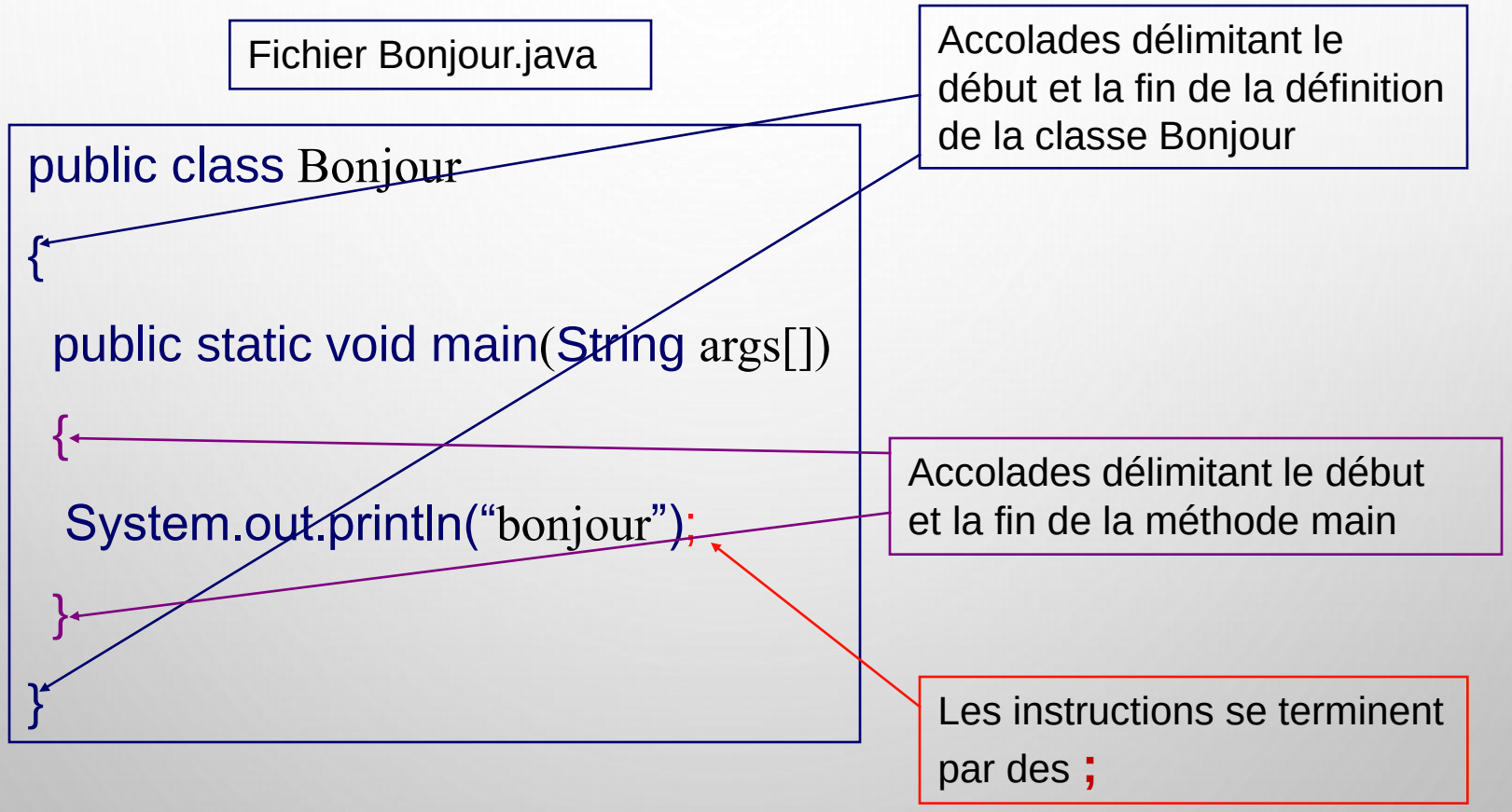
EXEMPLE (1)

Fichier Bonjour.java

```
public class Bonjour
{    //Accolade débutant la classe Bonjour
  public static void main(String args[])
  { //Accolade débutant la méthode main
    /* Pour l'instant juste une instruction */
   System.out.println("bonjour");
  } //Accolade fermant la méthode main
} //Accolade fermant la classe Bonjour
```

La classe est l'unité de base de nos programmes. Le mot clé en Java pour définir une classe est class

EXEMPLE (2)


Fichier Bonjour.java
public class Bonjour
{
public static void main(String args[])
{
System.out.println("bonjour");
}
}
Accolades délimitant le début et la fin de la définition de la classe Bonjour
Accolades délimitant le début et la fin de la méthode main
Les instructions se terminent par des ;

EXEMPLE (3)

Fichier Bonjour.java

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

La méthode **main** peut recevoir des paramètres. Ici la méthode main reçoit le paramètre args qui est un tableau de chaîne de caractères.

main : est le nom de la méthode
String : est le nom de la classe du JDK
args : est le nom du paramètre

COMPILATION ET EXÉCUTION (1)

Fichier Bonjour.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 Bonjour.java
Génère un fichier
Bonjour.class
Exécution du programme (toujours depuis la console DOS) sur la JVM :
java Bonjour
Affichage de « bonjour » dans la console

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

COMPILATION ET EXÉCUTION (2)

- Pour résumer, si on a un fichier Bonjour.java pour la classe Bonjour :
  - javac Bonjour.java
    - Compilation en bytecode java
    - Indication des erreurs de syntaxe éventuelles
    - Génération d'un fichier Bonjour.class si pas d'erreurs
  - java Bonjour
    - Java est la machine virtuelle
    - Exécution du bytecode
    - Nécessité de la méthode main, qui est le point d'entrée dans le programme

IDENTIFICATEURS (1)

- 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 distingue les majuscules des minuscules.
- Convention sur les identificateurs :
  - Si plusieurs mots sont accolés, mettre une majuscule à chacun des mots sauf le premier.
    - Exemple : uneVariableEntiere
  - La première lettre est majuscule pour les classes et les interfaces.
    - Exemples : MaClasse, UneJolieFenetre

IDENTIFICATEURS (2)

- Convention sur les identificateurs :
  - La première lettre est minuscule pour les méthodes, les attributs et les variables
    - Exemples : setLongueur, i, uneFenetre
  - Les constantes sont entièrement en majuscules
    - Exemple : LONGUEUR_MAX

LES MOTS RÉSERVÉS DE JAVA

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

LES TYPES DE BASES (1)

- 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 diverses tailles : 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 DE BASES (2) : 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 DE BASES (3) : LES ENTIERS

- Notation
  - 2 : entier normal en base décimal
  - 2L : entier au format long en base décimal
  - 010 : entier en valeur octale (base 8)
  - 0xF : entier en valeur hexadécimale (base 16)
- Opérations sur les entiers
  - opérateurs arithmétiques +, -, *
  - / :division entière si les 2 arguments sont des entiers
  - % : reste de la division entière
    - exemples :
      - 15 / 4 donne 3
      - 15 % 2 donne 1

LES TYPES DE BASES (4) : 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 utilise dans les expressions.
      En version suffixée la (déc/inc) rémentation s'effectue en premier.

```
int m=7; int n=7;
int a=2 * ++m; //a vaut 16, car m vaut 8
int b=2 * n++; //b vaut 14, et 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
  - 4.55F 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 (11) : 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 TYPES DE BASES (12)
INITIALISATION DES VARIABLES

- Voici les valeurs par défaut lors de l'initialisation automatique des variables d'instances sont :

Type	Valeur par défaut
boolean	false
byte, short, int, long	0
float, double	0.0
char	\u000
classe	null

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) { 1er bloc d'instructions}

    else {2ème bloc d'instruction}

LES STRUCTURES DE CONTRÔLES (3) : IF / ELSE

Max.java

```
public class Max
{
  public static void main(String args[])
  {
            int nb1 = 10;
            int nb2 =5;
     if (nb1 > nb2)
       System.out.println("l'entier le plus grand est "+ nb1);
     else
       System.out.println("l'entier le plus grand est "+ nb2);
   }
}
```

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.
  - En Java, il y a 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

Facto.java

```
public class Facto
{
  public static void main(String args[])
  {
     int n, result,i;
     n = 5;
     result = 1; i = n;
     while (i > 1)
       {
          result = result * i;
          i--;
       }
        System.out.println("la factorielle de "+n+" vaut "+result);
  }
}
```

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

```
public class Facto2
{
  public static void main(String args[])
  {
     int n, result,i;
     n = 5;
     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).
  - 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

```
public class Alternative
{
  public static void main(String args[])
  {
     int nb = 2;
     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.

LECTURE DES DONNÉES

LECTURE D'UNE DONNÉE (1)

- Pour créer des programmes interactifs acceptant les entrées d'un utilisateur, vous pouvez utiliser **System.in**, qui fait référence au périphérique d'entrée standard (généralement le clavier).
- L'objet **System.in** n'est pas aussi flexible que **System.out**; il est conçu pour ne lire que des octets. C'est un problème, car vous voulez souvent accepter des données d'autres types. Heureusement, les concepteurs de Java ont créé une classe nommée **Scanner** qui rend **System.in** plus flexible.
- Pour créer un objet Scanner et le connecter à l'objet **System.in**, vous écrivez une instruction similaire à celle-ci:

```
1Scanner clavier = new Scanner(System.in);
```

LECTURE D'UNE DONNÉE (2)

- La classe **Scanner** contient des méthodes permettant d'extraire des valeurs d'un périphérique d'entrée. Chaque valeur extraite est un jeton, qui est un ensemble de caractères séparé du prochain ensemble par des espaces.

- Le tableau suivant récapitule certaines des méthodes les plus utiles pour lire différents types de données à partir du périphérique d'entrée par défaut.

Méthode	Description
`nextDouble()`	Récupère l'entrée en tant que Double
`nextInt()`	Récupère l'entrée en tant que int
`nextLine()`	Récupère la ligne de données suivante et la renvoie sous forme de chaîne (String)
`next()`	Récupère le prochain jeton complet sous forme de chaîne
`nextShort()`	Récupère l'entrée en tant que Short
`nextByte()`	Récupère l'entrée en tant que octet (Byte)
`nextFloat()`	Récupère les entrées sous forme de float. Notez que lorsque vous entrez une valeur d'entrée qui sera stockée sous forme de float, vous ne tapez pas de F. Le F est utilisé uniquement avec les constantes codées dans un programme.
`nextLong()`	Récupère l'entrée en tant que long. Notez que lorsque vous entrez une valeur d'entrée qui sera stockée longtemps, vous ne tapez pas un L. Le L est utilisé uniquement avec des constantes codées dans un programme.

LES TABLEAUX EN JAVA

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 chaines[ ];
- Création d'un tableau
    - tab = new int [20]; // tableau de 20 int
    - chaines = 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.
  - La taille d'un tableau est récupérable par le mot clé **length** : 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 commence de 0 à 99.

LES TABLEAUX (3)

- Initialisation

```
tab[0]=1;
tab[1]=2; //etc.
noms[0] = new String( « Sam");
noms[1] = new String( "Bill");//etc
```

- Création et initialisation simultanées

```
String noms [ ] = {"Sam","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

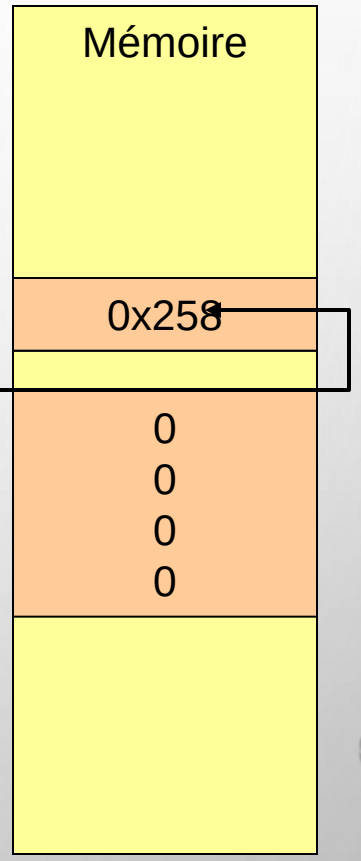
On peut ensuite affecter des valeurs au différentes cases du tableau : <nom_tableau>[indice]

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

LES TABLEAUX (5)

Tab1.java

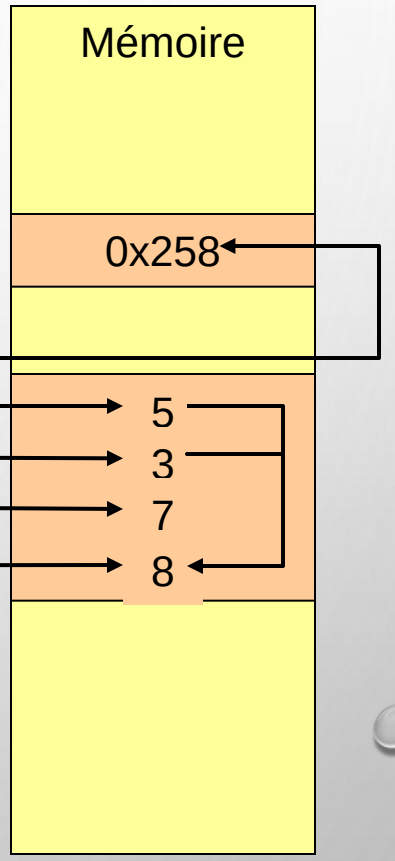
```
public class Tab1
{
  public static void main (String args[])
  {
    int tab[ ] ;
    tab = new int[4];
    tab[0]=5;
    tab[1]=3;
    tab[2]=7;
    tab[3]=tab[0]+tab[1];
  }
}
```



LES TABLEAUX (6)

Tab1.java

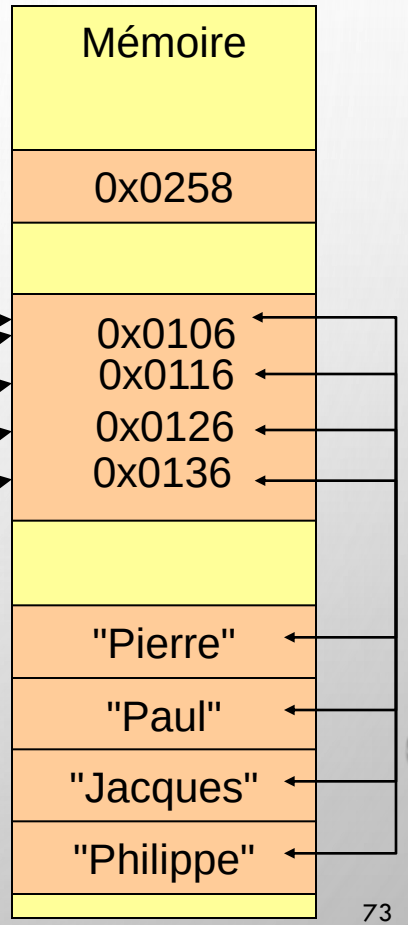
```
public class Tab1
{
  public static void main (String args[])
  {
    int tab[ ] ;
    tab = new int[4];
    tab[0]=5;
    tab[1]=3;
    tab[2]=7;
    tab[3]=tab[0]+tab[1];
  }
}
```



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 définie dans l'API Java (Dans le package java.lang)

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

  String s=new String("aaa"); // Permet de créer un objet de type String et lui affecter la chaine "aaa"

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

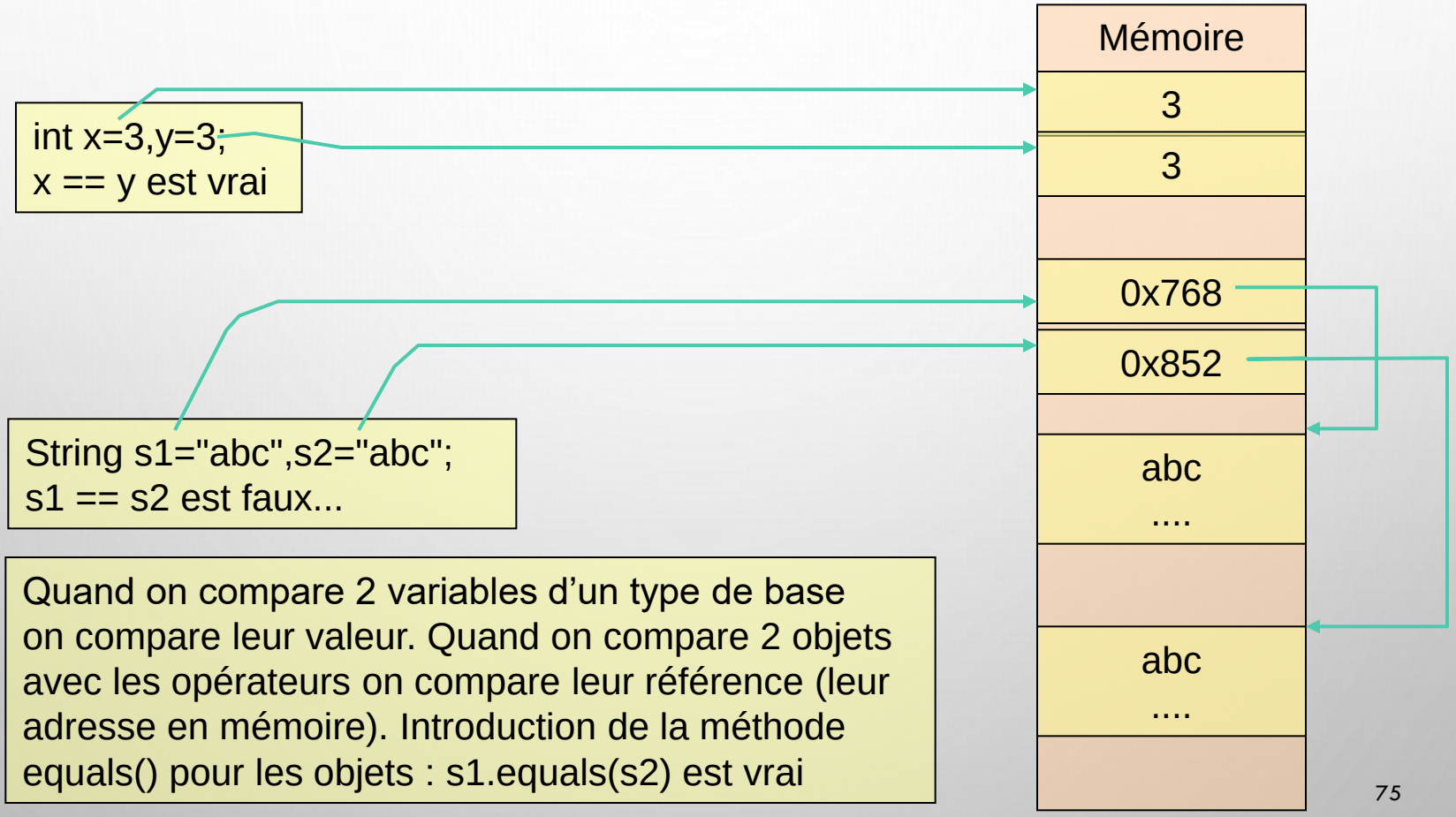
```
String str1 = "Bonjour ! ";

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



Quand on compare 2 variables d'un type de base on compare leur valeur. Quand on compare 2 objets avec les opérateurs on compare leur référence (leur adresse en mémoire). Introduction de la méthode equals() pour les objets : s1.equals(s2) est vrai

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-chaine 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 se message

```
String str1 = "bonjour";
char c = str1.charAt(3); // c 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).
  - pour tester si 2 chaînes sont égales à la casse, en ignorant les majus et les minus, 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 true
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 chaine de caractère commence par la chaine de caractère str
  - boolean endsWith(String str) : pour tester si une chaîne de caractère se termine par la chaine de caractère str

```
String str1 =   "bonjour ";
boolean a = str1.startsWith("bon");//a vaut true
boolean b = str1.endsWith("jour");//b vaut true
```

LA CLASSE STRING (7)

Plus d'informations dans les documentations de l'API dans le Package : **java.lang**

LA CLASSE MATH

- Les fonctions mathématiques les plus connues sont regroupées dans la classe **Math** qui appartient au package java.lang
  - les fonctions trigonométriques
  - les fonctions d'arrondi, de valeur absolue, ...
  - la racine carrée, la puissance, l'exponentiel, le logarithme, etc.
- Ce sont des méthodes de classe (static)

```
double calcul = Math.sqrt (Math.pow(5,2) + Math.pow(7,2));
```

double **sqrt**(double x) : racine carrée de x

double **pow**(double x, double y) : x puissance y

TO BE CONTINUED !