Programmation Orientée Objet (POO)

Licence en Informatique

Langage Java

papaddiop@gmail.com
Prof. Papa DIOP – UFR SET - Université de THIES

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*)

Historique de Java (3)

- Les développeurs Java ont réalisé un langage indépendant de toute architecture de telle sorte que Java devienne idéal pour programmer des applications utilisables dans des réseaux hétérogènes, notamment Internet.
- Le développement de Java devint alors un enjeu stratégique pour **Sun** et l'équipe écrivit un navigateur appelé **HotJava** capable d'exécuter des programmes Java.
- 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
  - 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 5 en 2004
  - Java 6 en 2006 *celle que nous utiliserons dans ce cours*
  - Java 7 en 2011
- 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
  - avec certains navigateurs

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: faille de sécurité importante dans Java 7

Que penser de Java? (1)

- Les plus :
  - Il a su bénéficier de l'essor d'Internet
  - Il a su s'imposer dans de nombreux domaines
  - Un environnement gratuit et de nombreux outils disponibles
  - Une large communauté très active

Que penser de Java? (2)

- Les moins :
  - Trop '*médiatisé*'?
  - Problèmes de compatibilité
    - Avec les premières versions
    - Avec certains navigateurs (les navigateurs ne sont pas écrits par Sun)
  - Problèmes de vitesse, mais existence de solutions pour y pallier (compilateur natif, compilation du bytecode à la volée)

Que penser de Java? (3)

- Un bon langage
- Un langage adapté aux besoins de développements actuels …
- … qui a su se baser sur les acquis du passé.
- Au delà de l'effet de mode, un succès mérité qui ne devrait pas se démentir dans le futur

Organisation du cours (1)

- Nous verrons :
  - Caractéristiques de Java et son environnement de développement
  - Structures fondamentales
  - La programmation par objets en Java
    - Héritage
    - Polymorphisme
  - 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 applets
  - Les threads (appelés aussi *processus légers*)
  - La programmation réseau
  - 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
    - JBuilder 5 : http://www.borland.fr/download/jb5pers/
  - 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 *(cela semble évident, mais pourtant!)*
  - avant de construire un bâtiment, on fait un plan. Ce n'est pas différent en informatique : conception de l'algorithme, i.e. une réponse (rationnelle) au problème posé
  - puis mise en œuvre technique - le *codage* - dans un langage de programmation, dans notre cas Java.

Analyse du problème (1)

- Se poser les bonnes questions
  - Quelles sont les objets 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)

- Savoir être :
  - **efficace** : quelle méthode me permettra d'obtenir le plus vite, le plus clairement, le plus simplement possible les résultats attendus ?
  - **paresseux** : dans ce que j'ai développé avant, que puis-je réutiliser ?
  - **prévoyant** : comment s'assurer que le programme sera facilement réutilisable et extensible ?

Caractéristiques du langage Java

Caractéristiques du langage Java (1)

- Simple
  - Apprentissage facile
    - faible nombre de mots-clés
    - simplifications des fonctionnalités essentielles
  - Développeurs opérationnels rapidement

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

Caractéristiques du langage Java (2)

- Orienté objet
  - Java ne permet d'utiliser 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
- 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 du langage Java (3)

- Fiable
  - Gestion automatique de la mémoire *(ramasse-miette ou "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 de programmation

- Applications Java : programmes autonomes, "stand-alone"
- Applets (mini-programmes) : Programmes exécutables uniquement par l'intermédiaire d'une autre application
  - navigateur web : Netscape, Internet explorer, Hotjava
  - application spécifique : Appletviewer
- Java est souvent considéré comme étant uniquement un langage pour écrire des applets alors que c'est aussi un vrai langage de programmation
- Java est souvent confondu avec le langage de script Javascript auquel il n'est en aucune manière apparenté

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)

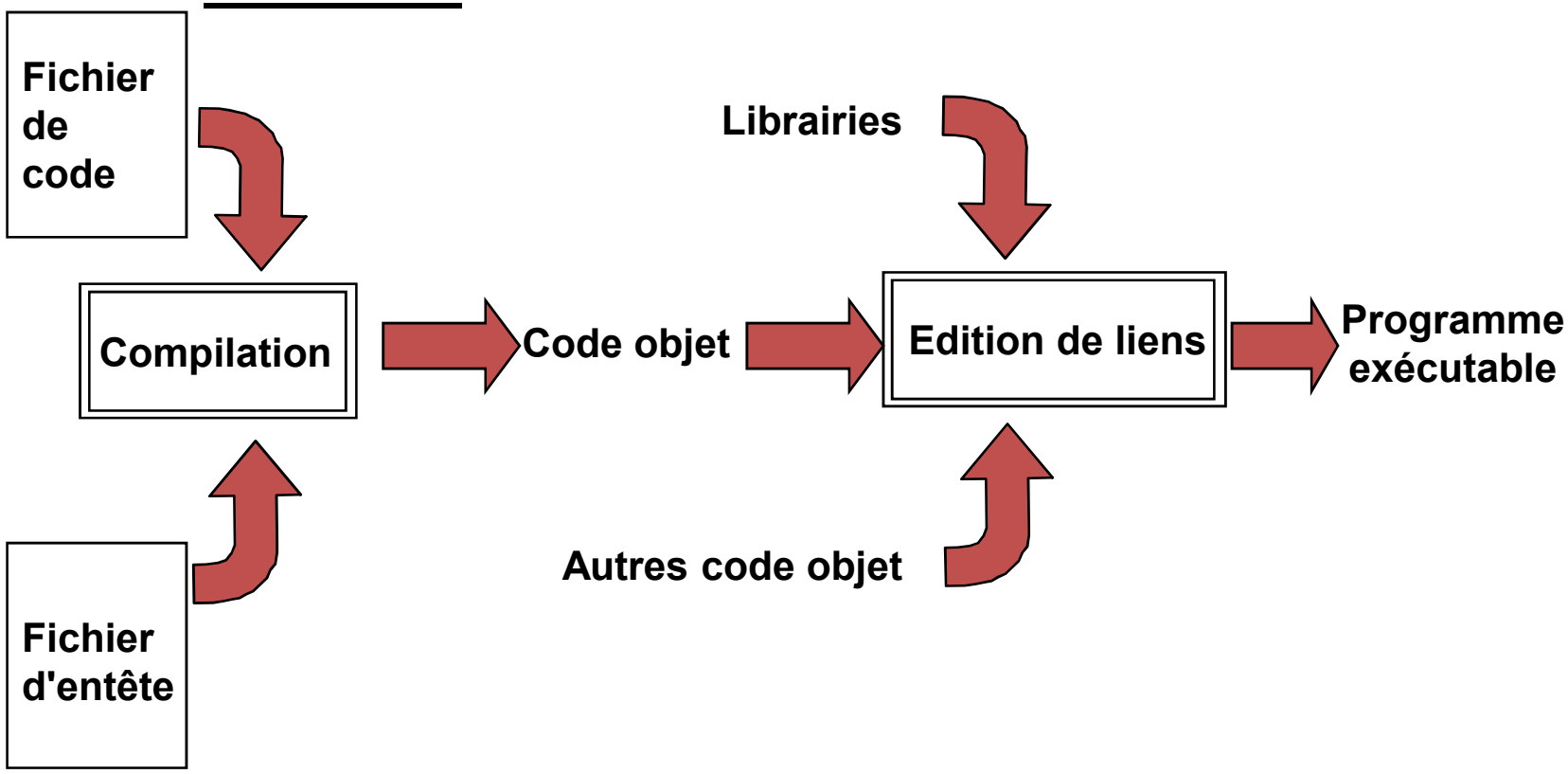
- Avantages :
  - **Portabilité**
    - Des machines virtuelles Java existent pour de nombreuses plates-formes dont : Linux, Windows, MacOS
  - **Développement plus rapide**
    - courte étape de compilation pour obtenir le byte-code,
    - pas d'édition de liens,
    - déboguagge plus aisé,
  - **Le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

Java, un langage indépendant? (3)

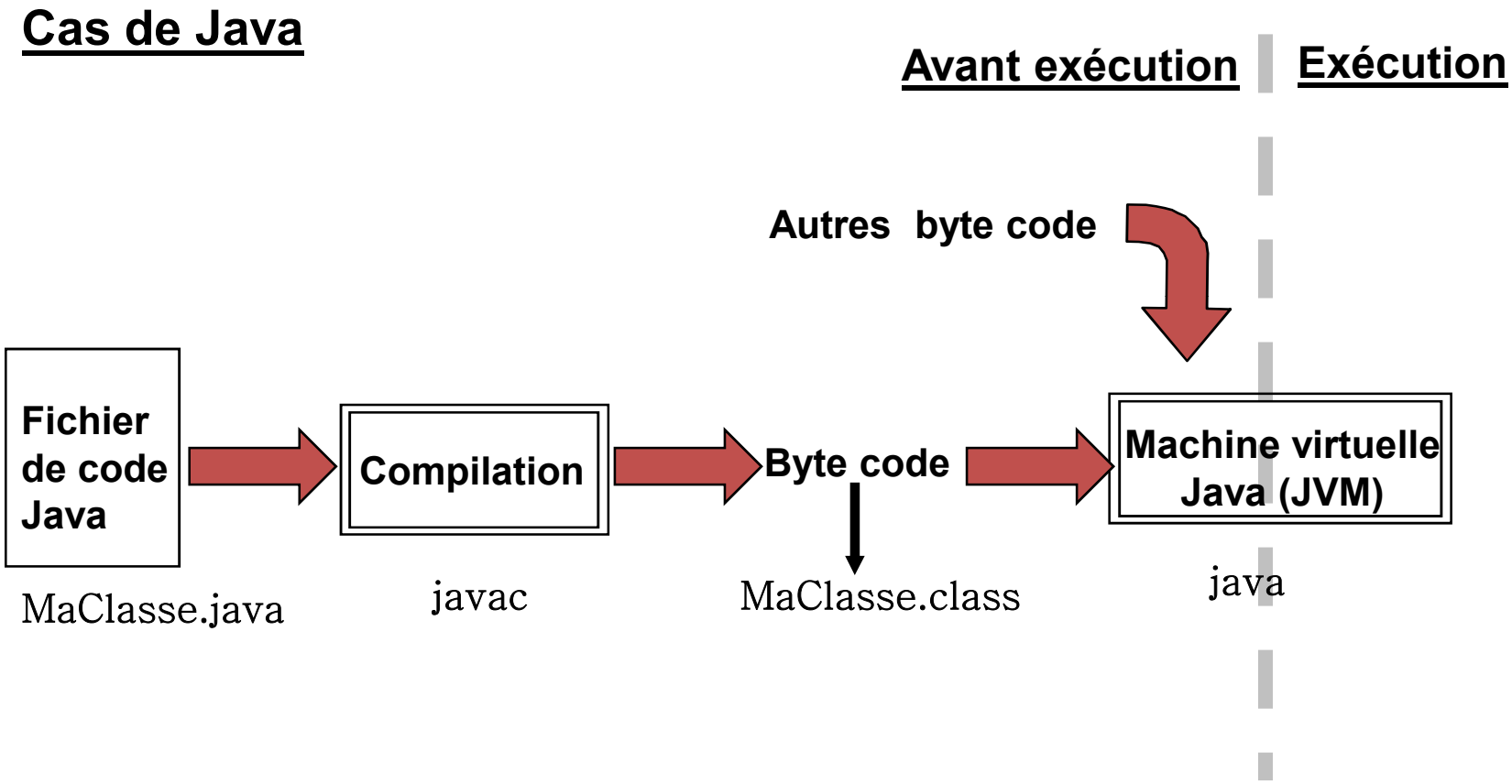
- Inconvénients :
  - Nécessite l'installation d'un interpréteur pour pouvoir exécuter un programme Java
  - L'interprétation du code ralentit l'exécution
  - Les applications ne bénéficient que du dénominateur commun des différentes plate-formes
    - limitation, par exemple, des interfaces graphiques
  - 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



L’API de Java

- 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 (pas abordé dans ce cours)
  - java.sql : fournit le package JDBC (pas abordé dans ce cours)

L’API de Java (3)

- 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.
  - intérêt de l'hypertexte pour naviguer dans la documentation

L'API de Java (4)

- 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)

- 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.

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[ ])
{
.../...
}
```

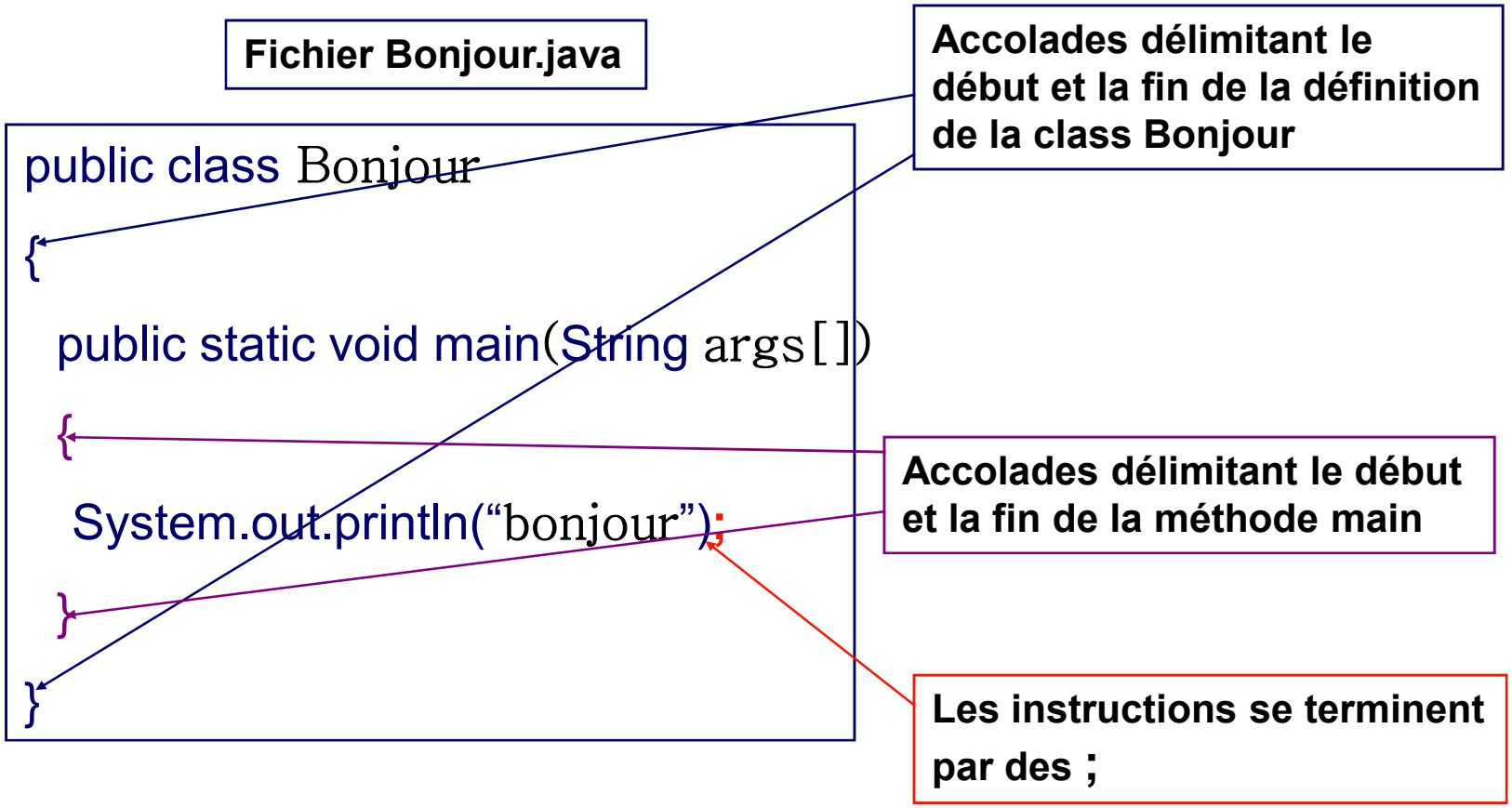
Exemple (1)

Fichier 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 class 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");
  }
}
```

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

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, dans une console DOS, si j'ai 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
- Conventions 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)

- Conventions 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 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 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
  - **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
  - % : 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 et **--**
    - 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 suffixée 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 (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 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

```
import java.io.*;

public class Max
{
  public static void main(String args[])
  {
            Console console = System.console();
            int nb1 = Integer.parseInt(console.readLine("Entrer un entier:"));
            int nb2 = Integer.parseInt(console.readLine("Entrer un autre entier:"));
     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
  - 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

Facto1.java

```
import java.io.*;

public class Facto1
{
  public static void main(String args[])
  {
     int n, result,i;
     n = Integer.parseInt(System.console().readLine("Entrer une valeur pour n:"));
     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

```
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**).
  - 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  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. 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

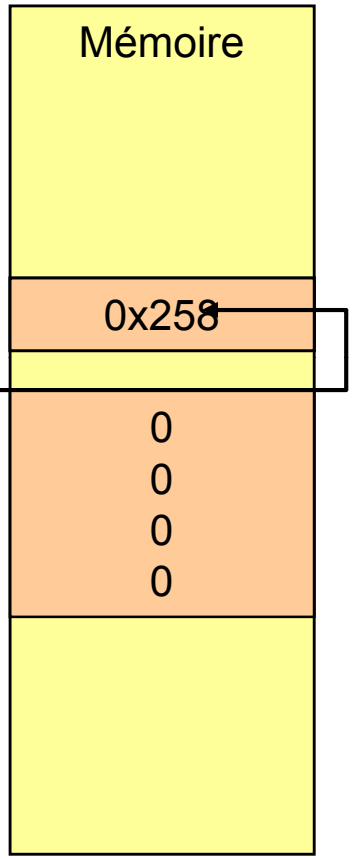
On peut ensuite affecter des valeurs au différentes cases du tableau : <nom_tableau>[indice]

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

Les tableaux (5)

Tab1.java

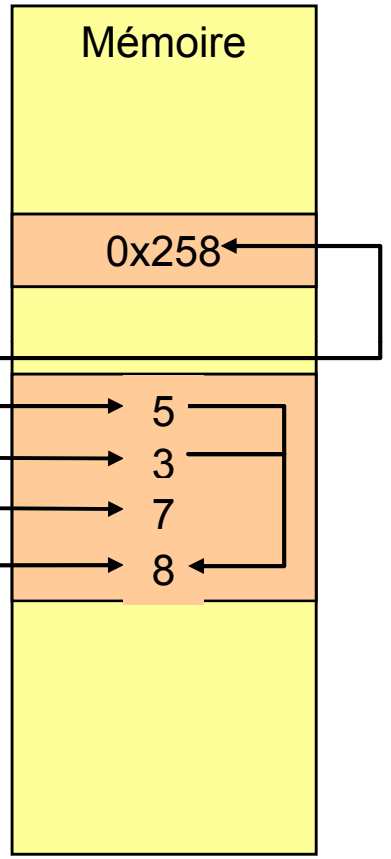
```
public class Tab1
{
  public static void main (String args[])
  {
    int tab[ ] ;
    tab = new int[4];
    tab[0]=5;
    tab[1]=3;
    tab[2]=7;
    tab[3]=tab[0]+tab[1];
  }
}
```



Les tableaux (6)

Tab1.java

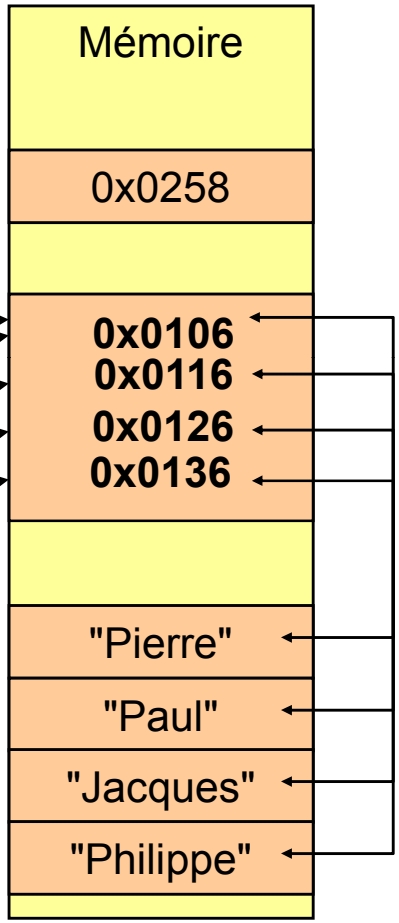
```
public class Tab1
{
  public static void main (String args[])
  {
    int tab[ ] ;
    tab = new int[4];
    tab[0]=5;
    tab[1]=3;
    tab[2]=7;
    tab[3]=tab[0]+tab[1];
  }
}
```



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

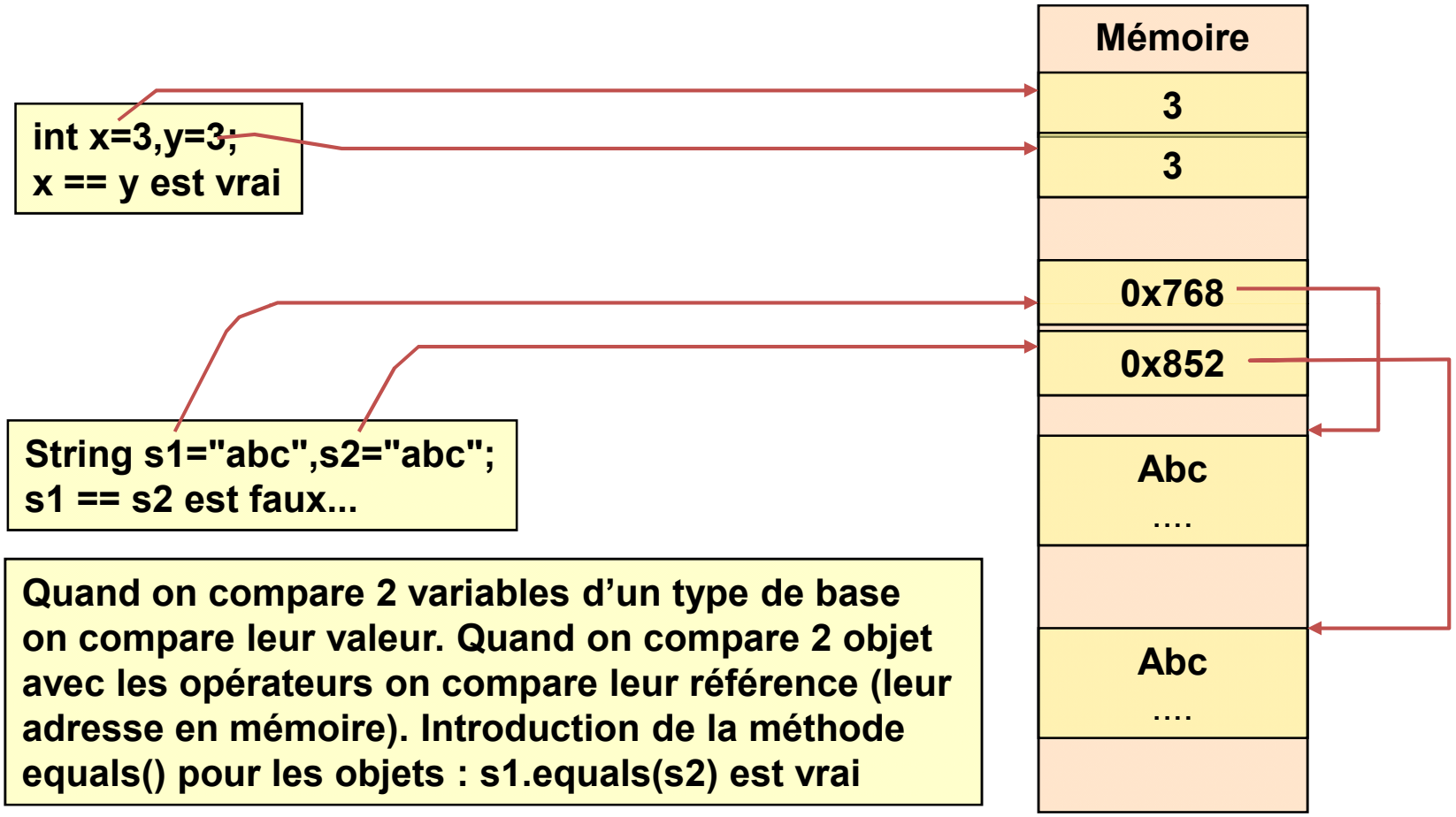
  String s="aaa"; // s contient la chaîne "aaa" mais

  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-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 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 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 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

La classe java.util.Scanner

- La classe Scanner permet de lire un flux de données formatées.

- *Exemple*

```
public static void main(String arg[])
{
    Scanner entree = new Scanner(System.in);
    System.out.println("Donner votre prenom et votre nom:");
    String prenom = entree.next(); String nom = entree.next();
    System.out.println("Donner votre age:");
    int age = entree.nextInt();
    entree.nextLine();
    System.out.println("Donner une phrase:");
    String phrase = entree.nextLine();
    System.out.printf("%s %s, %d ans, dit
    %s\n",prenom,nom,age,phrase);
}
```

La classe java.util.Scanner

• Offre des méthodes pour lire simplement un flux de données formatées :

☞ int nextInt() : lecture de la prochaine valeur de type int.

☞ long nextLong() : lecture de la prochaine valeur de type long.

☞ float nextFloat() : lecture de la prochaine valeur de type float.

☞ double nextDouble() : lecture de la prochaine valeur de type double.

☞ String nextLine() : lecture de tous les caractères se présentant dans le flux d'entrée jusqu'à une marque de fin de ligne et renvoie de la chaîne ainsi construite. La marque de fin de ligne est consommée, mais n'est pas incorporée à la chaîne produite.

La classe Scanner

• Offre aussi toute une série de méthodes booléennes pour tester le type de la prochaine donnée disponible, sans lire cette dernière :

- ✓ boolean hasNext() : Y a-t-il une donnée disponible pour la lecture
- ✓ boolean hasNextLine() : Y a-t-il une ligne disponible pour la lecture
- ✓ boolean hasNextInt() : La prochaine donnée à lire est-elle de type int
- ✓ boolean hasNextLong() : La prochaine donnée à lire est-elle de type long
- ✓ boolean hasNextFloat() : La prochaine donnée `à lire est-elle de type float
- ✓ boolean hasNextDouble() : La prochaine donn´ee `a lire est-elle de type double