Le langage JAVA

Les bases : la syntaxe

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[ ])
{
…/…
}
```

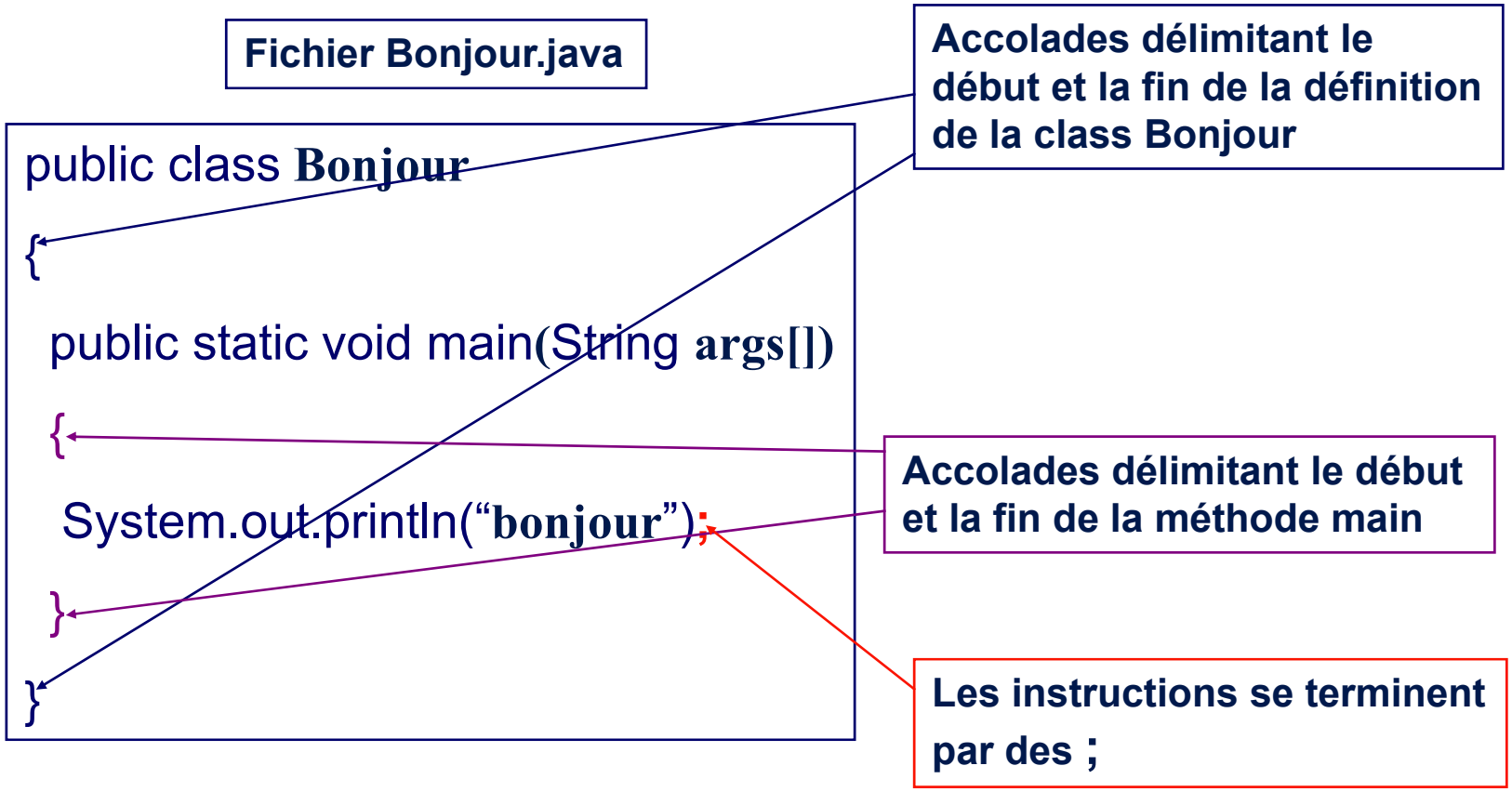
EXEMPLE (1)

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 (3)

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
- 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 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.
- Chaque type de base possède son objet associé avec des méthodes pour le manipuler
  Exemple : Integer pour int, Boolean pour boolean (Attention à la majuscule)

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 {
        // Lecture sur la console Windows ou la console Eclipse
        private static String readLine(String prompt) {
          String line = null;
          Console c = System.console();
          if (c != null) {                          // la console répond ➔ on l'utilise
              line = c.readLine(prompt);            // on lit sur la console
          } else {                                  // sinon on utilise le BufferedReader pour lire
              System.out.print(prompt);
              BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));
              try {
                  line = bufferedReader.readLine();
              } catch (IOException e) {
                        System.out.print("Erreur de saisie");
              }
          }
          return line;
        }

        public static void main(String[] args)  {
              int nb1 = Integer.parseInt(readLine("Entrer un entier:"));
              int nb2 = Integer.parseInt(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;
    Scanner entree = new Scanner(System.in); // utilisation d'une autre classe de lecture
    System.out.println("Entrer une valeur pour n:");
    n = entree.nextInt();
    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;
    Scanner entree =   new Scanner(System.in);
    System.out.println("Entrer une valeur pour n:");
    n = entree.nextInt();
     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[]) {
     Scanner entree =   new Scanner(System.in);
     System.out.println("Entrer une valeur pour n : ");
     int nb = entree.nextInt();
     switch(nb)
     {
       case 1:
         System.out.println("Un"); break;
       case 2:
         System.out.println("Deux"); break;
       default:
        System.out.println("Autre nombre");
     }
  }
```

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 entiers*
- *chaines = new String [100]; // tableau de 100 chaînes*

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.

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

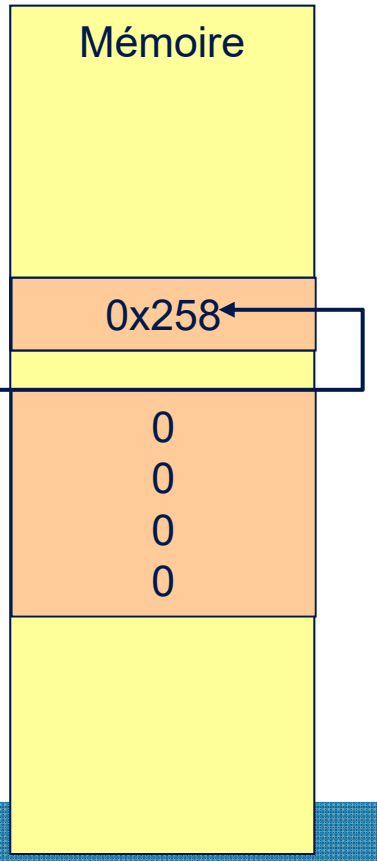
On peut ensuite affecter des valeurs au différentes cases du tableau : <nom_tableau>[indice]

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

LES TABLEAUX (5)

Tab1.java

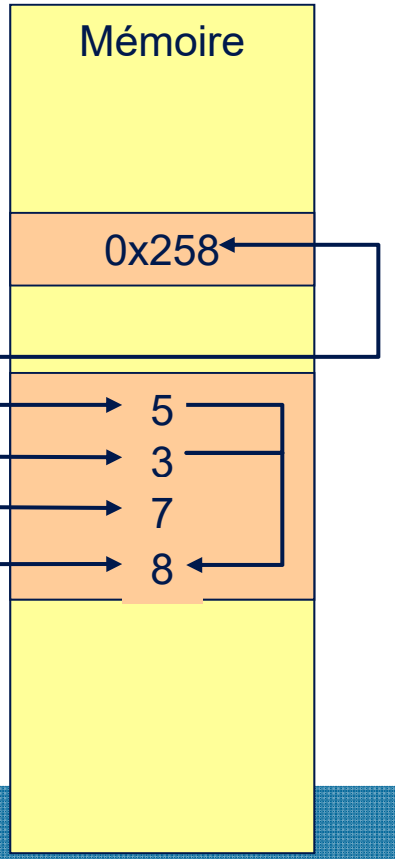
```
public class Tab1
{
  public static void main (String args[])
  {
    int tab[ ] ;
    tab = new int[4];
    tab[0]=5;
    tab[1]=3;
    tab[2]=7;
    tab[3]=tab[0]+tab[1];
  }
}
```



LES TABLEAUX (6)

Tab1.java

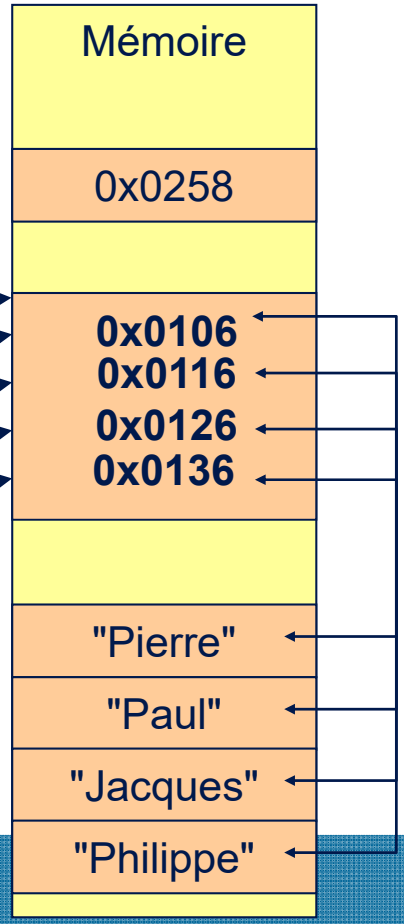
```
public class Tab1
{
  public static void main (String args[])
  {
    int tab[ ] ;
    tab = new int[4];
    tab[0]=5;
    tab[1]=3;
    tab[2]=7;
    tab[3]=tab[0]+tab[1];
  }
}
```



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)

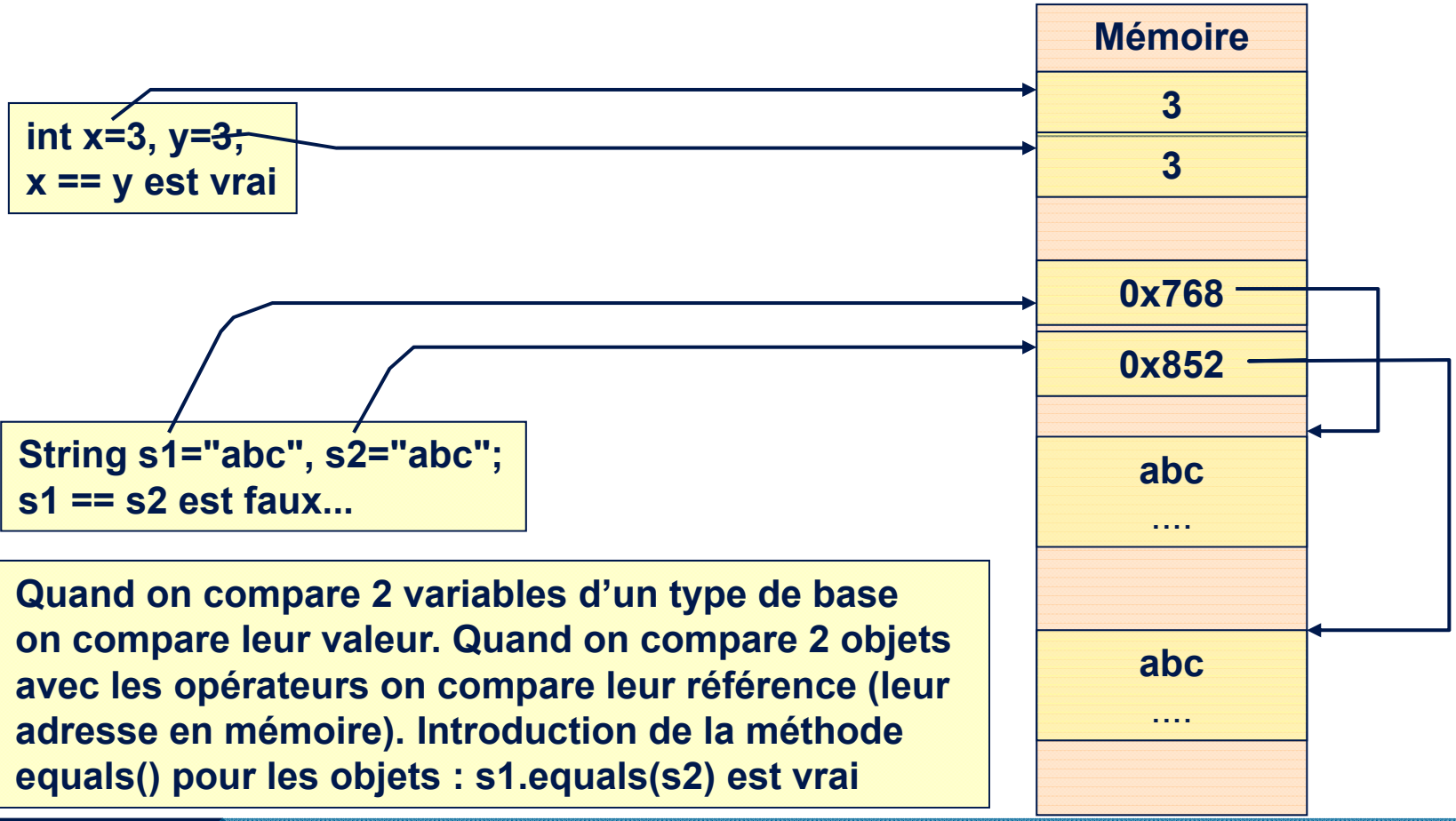
```
String s = "aaa";                  // s contient la chaîne "aaa" mais
String s = new String("aaa");  // identique à la ligne précédente
```

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



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 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)

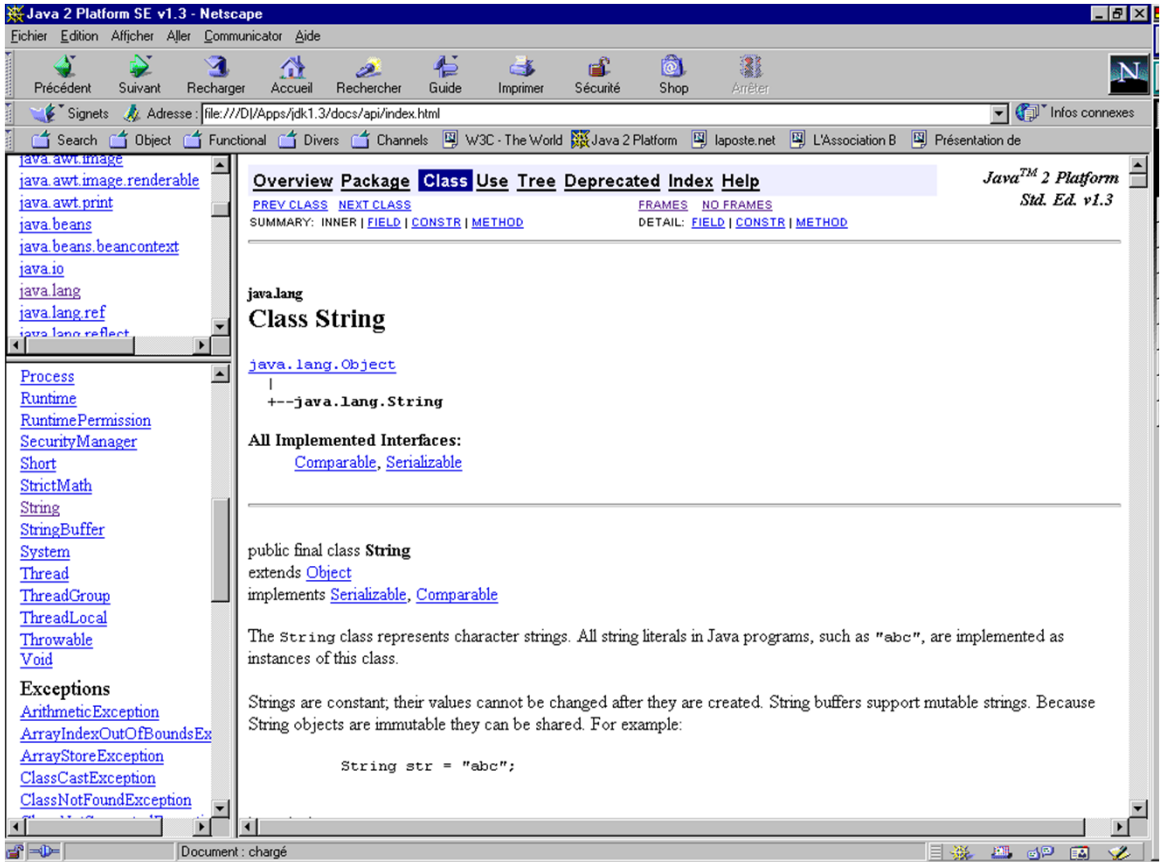
Quelques autres méthodes utiles

- *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 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*