Types et instructions de base Java

Certains de ces transparents sont une reprise des transparents du cours *"Comment JAVA ? Java bien !"* de P. Itey

Identificateurs

- Nommer les classes, les variables, les méthodes, ...
- Un identificateur Java
  - *est de longueur quelconque*
  - *commence par une lettre Unicode (caractères ASCII recommandés)*
  - *peut ensuite contenir des lettres ou des chiffres ou le caractère souligné « _ »*
  - *ne doit pas être un mot réservé du langage (mot clé) (**`if, for, true,`** …)*

```
[a..z, A..Z, $, _]{a..z, A..Z, $, _, 0..9, Unicode}
```

Conventions pour les identificateurs

- Les noms de classes commencent par une majuscule (ce sont les seuls avec les constantes) :
  - **`Visage, Object`**

- Les mots contenus dans un identificateur commencent par une majuscule :
  - **`UneClasse, uneMethode, uneVariable`**
  - On préférera **`ageDuCapitaine`** à **`ageducapitaine`** ou **`age_du_capitaine`**

- Les constantes sont en majuscules et les mots sont séparés par le caractère souligné « _ » :
  - **`UNE_CONSTANTE`**

Mots-clés Java

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

Commentaires

- Sur une ligne

```
// Comme en “C++”, après un slash-slash
int i; // commentaire jusqu’à la fin de la ligne
```

- Sur plusieurs lignes

```
/* Comme en “C”, entre un slash-étoile et
   un étoile-slash, sur plusieurs lignes */
```

- Commentaires documentant pour l’outil **javadoc**

```
/**
 * pour l’utilisation de Javadoc
 * à réserver pour la documentation automatique
 * avec javadoc
 */
```

Commentaires

- Commenter le plus possible et judicieusement
- Commenter clairement (utiliser au mieux les 3 possibilités)
- Chaque déclaration de classe et de membre d'une classe (variable et méthode) **doit** être commentée avec un commentaire javadoc /** ... */

Types de données en Java

- 2 grands groupes de types de données :
  - *types primitifs*
  - *objets (instances de classe)*
- Java manipule différemment les valeurs des types primitifs et les objets : les variables contiennent
  - *des valeurs de types primitifs*
  - *ou des références aux objets*

Types primitifs

- Valeur logique
  - ***boolean*** *(true/false)*
- Nombres entiers
  - ***byte*** *(1 octet),* ***short*** *(2octets),* ***int*** *(4 octets),* ***long*** *(8 octets)*
- Nombres non entiers (à virgule flottante)
  - ***float*** *(4 octets),* ***double*** *(8 octets).*
- Caractère (un seul)
  - ***char*** *(2 octets) ; codé par le codage Unicode (et pas ASCII)*

- types indépendants de l'architecture
  - *En C/C++, représentation dépendante de l'architecture (compilateur, système d'exploitation, processeur)*
    ex: int = 32 bits sous x86, mais 64 bits sous DEC alpha
    Portage difficile, types numériques signés/non signés

Types primitifs et valeurs

Type	Taille	Valeurs
`boolean`	1	`true`, `false`
`byte`	8	-2^7 à $+2^7-1$
`char`	16	0 à 65535
`short`	16	-2^{15} à $+2^{15}-1$
`int`	32	-2^{31} à $+2^{31}-1$
`long`	64	-2^{63} à $+2^{63}-1$
`float`	32	1.40239846e-45 à 3.40282347e38
`double`	64	4.94065645841246544e-324 à 1.79769313486231570e308

Constantes nombres

- Une constante «entière» est de type **long** si elle est suffixée par «**L**» et de type **int** sinon
- Une constante «flottante» est de type **float** si elle est suffixée par «**F**» et de type **double** sinon

- **Exemples**
  - `35`
  - `2589L // constante de type long`
  - `0.5 // de type double`
  - `4.567e2 // 456,7 de type double`
  - `0.5f  // de type float`
  - `.123587E-25F // de type float`

Constantes de type caractère

- Un caractère Unicode entouré par 2 simples quotes "'"
- Exemples :
  - `'A' 'a' 'ç' '1' '2'`
  - `\` *caractère d'échappement pour introduire les caractères spéciaux*
    - `'\t'` tabulation
    - `'\n'` nouvelle ligne
    - `'\r'` retour chariot, retour arrière
    - `'\f'` saut de page
    - …
    - `'\\' '\'' '\"'`
  - *'* `\u03a9` *' (* `\u` *suivi du code hexadécimal à 4 chiffres d'un caractère Unicode)*
    - `'α'`

Autres constantes

- Type booléen
  - `false`
  - `true`
- Référence inexistante (indique qu'une variable de type non primitif ne référence rien)
  - `null`

Forcer un type en Java

- Java langage fortement typé
  - *le type de donnée est associé au nom de la variable, plutôt qu'à sa valeur. (Avant de pouvoir être utilisée une variable doit être déclarée en associant un type à son identificateur).*
  - *la compilation ou l'exécution peuvent détecter des erreurs de typage*

- Dans certains cas, nécessaire de forcer le compilateur à considérer une expression comme étant d'un type qui n'est pas son type réel ou déclaré
  - *On utilise le* cast ou transtypage*:* **(**type-forcé**)** expression

- **Exemple**
  - ***int i = 64;***
  - ***char c = (char)i;***

Casts entre types primitifs

- Un *cast* entre types primitifs peut occasionner une perte de données
  - *Par exemple, la conversion d'un **int** vers un **short** peut donner un nombre complètement différent du nombre de départ.*

```
int i = 32768;
short s = (short) i;
System.out.println(s);   → -32767;
```

- Un *cast* peut provoquer une simple perte de précision
  - *Par exemple, la conversion d'un **long** vers un **float** peut faire perdre des chiffres significatifs mais pas l'ordre de grandeur*

```
long l1 = 928999999L;
float f = (float) l1;
System.out.println(f); → 9.29E8
long l2 = (long) f;
System.out.println(l2); → 929000000
```

Casts entre types primitifs

- Les affectations entre types primitifs peuvent utiliser un *cast* implicite si elles ne peuvent provoquer qu'une perte de précision (ou, encore mieux, aucune perte)

```
int i = 130;
double x = 20 * i;
```

- Sinon, elles doivent comporter un *cast* explicite

```
short s = 65; // cas particulier affectation int "petit"
s = 1000000; // provoque une erreur de compilation
int i = 64;
byte b = (byte)(i + 2); // b = 66
char c = (char) i; // caractère dont le code est 64 → '@'
b = (byte)128; // b = -128 !
```

Type mismatch: cannot convert from int to short

Casts entre entiers et caractères

- La correspondance **char** → **int**, **long** s'obtient par *cast* implicite

```
char c = '@',
int i = c;                    // ⇔int i = (int) c;
System.out.println(i);        // → 64 le rang du
                              //caractère '@'dans le codage
```

- Les correspondances **char** → **short**, **byte** , et **long**, **int**, **short** ou **byte** → **char** nécessitent un *cast* explicite (entiers sont signés et pas les **char**)

```
char c = '@';
short s = c;   Type mismatch: cannot convert from char to short   short s = (short) c;
int i = 64;
char c = i;    Type mismatch: cannot convert from int to char     char c = (char) i;
```

Opérateurs

- Les plus utilisés
  - *Arithmétiques*
    - `+ - * /`
    - `%` **(modulo)**
    - `++ --` (pré ou post décrémentation)
  - *Logiques*
    - `&&` (et) `||` (ou) `!` (négation)
  - *Relationnels*
    - `== != < > <= >=`
  - *Affectations*
    - `= += -= *=` ...

Ordre de priorité des opérateurs

Postfixés	`[] . (params) expr++ expr--`
Unaires	`++expr --expr +expr -expr ~ !`
Création et cast	`new (type)expr`
Multiplicatifs	`* / %`
Additifs	`+ -`
Décalages bits	`<< >>`
Relationnels	`< > <= >= instanceof`
Egalité	`== !=`
Bits, et	`&`
Bits, ou exclusif	`^`
Bits, ou inclusif	`|`
Logique, et	`&&`
Logique, ou	`||`
Conditionnel	`? :`
Affectation	`= += -= *= /= %= &= ^= |= <<= >>=`

Les opérateurs d'égales priorités sont évalués de gauche à droite, sauf les opérateurs d'affectation, évalués de droite à gauche

Déclarations

- Avant toute utilisation dans un programme une variable doit être déclarée
- syntaxe: ***type identificateur***
  - type : un type primitif ou un nom de classe
- Exemples

```
byte age;
boolean jeune;
float poids;
double x, y ,z;
```

- Une variable est accessible (**visible**) depuis l'endroit où elle est déclarée jusqu'à la fin du bloc où sa déclaration a été effectuée

Affectation

- Syntaxe : *lvalue* **=** *expression*

  lvalue est une expression qui doit délivrer une variable (par exemple un identificateur de variable, élément de tableau...., mais pas une constante)

- Exemples

```
int age;
age = 10;
boolean jeune = true;
float poids = 71.5f;
float taille = 1.75f;
float poidsTaile = poids / taille;
```

- Attention en JAVA comme en C, l'affectation est un opérateur. L'affectation peut donc être utilisée comme une expression dont la valeur est la valeur affectée à la variable

```
i = j = 10;
```

bloc d'instructions - instruction composée

- *permet de grouper un ensemble d'instructions en lui donnant la forme syntaxique d'une seule instruction*

- syntaxe:

{
séquence d'énoncés
}

- exemple

```
int k;
{
        int i = 1;
        int j = 12;
        j = i+1;
        k = 2 * j - i;
}
```

Instruction conditionnelle - instruction **if**

- Syntaxe

if (*expression booléenne* **)** *instruction1*

ou bien

if (*expression booléenne* **)**
instruction1
else
instruction2

- exemple

```
if (i==j){
   j = j -1;
   i = 2 * j;
}
else
   i = 1;
```

Un bloc car *instruction1* est composée de deux instructions

boucle tantque ... faire - instruction **while ()**

- Syntaxe **while (** *expression booléenne* **)**
  instruction

- Exemple

```
int i = 0;
int somme = 0;
while (i <= 10){
    somme += i;
    i++;
}
System.out.println("Somme des 10 premiers entiers" + somme);
```

boucle répeter ... jusqu'à – instruction do while ()

- Syntaxe

do
instruction
while (*expression booléenne* **) ;**

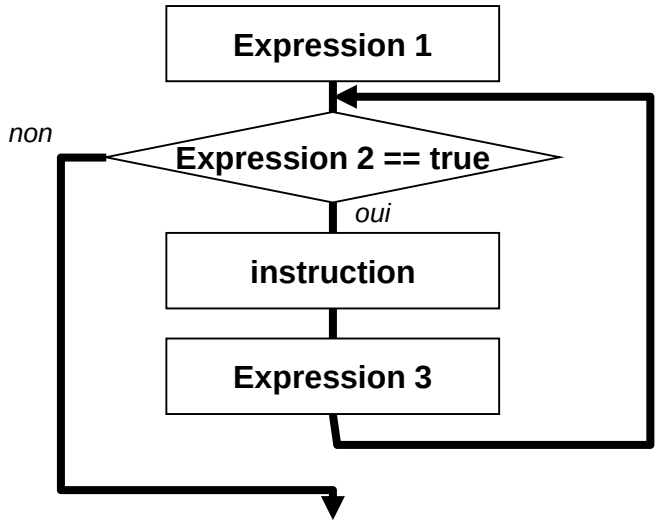
- Exemple

```
int i = 1 n;
int somme = 0;
do
{
    somme += i;
    i++;
}  while (i <= 10);
System.out.println("Somme des 10 premiers entiers" + somme);
```

boucle pour – instruction for

- Syntaxe **for (***expression1* **;** *expression2***;** *expression3***)**
  instruction

- Exemple




```
int  i;
int somme = 0;
for (i = 0; i <= 10; i++)
   somme += i;

System.out.println("Somme des 10 premiers entiers" + somme);
```

Affichage sur la console

System.out.print(*chaîne de caractères à afficher***)**
System.out.println(*chaîne de caractères à afficher***)**

- chaîne de caractères peut être :
  - *une constante chaîne de caractères (String)*
    ```
    System.out.println("coucou");
    ```
  - *une expression de type String*
    ```
    System.out.println(age);
    ```
    Ici `age` est une variable de type **int**
    Elle est **automatiquement** convertie en **String**
  - *une combinaison (concaténation) de constantes et d'expressions de type String. La concaténation est exprimée à l'aide de l'opérateur +*
    ```
    System.out.println("L'age de la personne est " +
        age + " son poids " + poids);
    ```
    `age` (int) et `poids` (float) sont **automatiquement** converties en **String**

Entrées/sorties sur console

- Lecture de valeurs au clavier

  - *classe* **`LectureClavier`** *facilitant la lecture de données à partir du clavier. Définit une méthode de lecture pour les types de base les plus couramment utilisés (int, float, double, boolean, String)*

```
System.out.print("entrez un entier : ");
int i = LectureClavier.lireEntier();
System.out.println("entier lu : " + i);

String s = LectureClavier.lireChaine("entrez une chaine : ");
System.out.println("chaine lue : " + s);

double d = LectureClavier.lireDouble("entrez un réel (double) : ");
System.out.println("réel (double) lu : " + d);


boolean b = LectureClavier.lireOuiNon("entrez une réposne O/N : ");
System.out.println("booleen lu : " + b);
```

DEPRECATED

`LectureClavier` n'est pas une classe standard de java. Pour l'utiliser vous devrez la récupérer sur le site Web de cet enseignement et l'intégrer à vos programme. Sa raison d'être est que dans les versions initiales de Java il n'y avait pas de moyen "simple" de faire ces opérations. Ce n'est plus le cas, depuis la version 5 de Java et l'introduction de la classe `Scanner` (du package java.util).

Entrées/sorties sur console

- Lecture de valeurs au clavier

  - *utiliser la classe `Scanner` du package standard `java.util` (version JDK 1.5 et supérieur).*

  - *Dans le fichier `ScannerDemo.java`*
    (http://lig-membres.imag.fr/genoud/ENSJAVA/tds/sujets/td2/src/ScannerDemo.java)
    vous trouverez un exemple d'utilisation de cette classe pour lire des données au clavier.

Mon premier programme Java

Le code de la classe doit être enregistré dans un fichier de même nom (casse comprise) que la classe

`HelloWorld.java`

(3)

(1) Tout code java doit être défini à l'intérieur d'une classe

```
public class HelloWorld {

    public static void main(String[] args) {

        int i = 0;

        while (i < 5) {
            System.out.println("Hello World !");
            i++;
        }

    }
}
```

Le point d'entrée pour l'exécution est la méthode `main()`

(4)

(2) La description de la classe est effectuée à l'intérieur d'un bloc { }

Compilation :

`javac HelloWorld.java`

`HelloWorld.java`

javac

`HelloWorld.class`

Exécution :

`java HelloWorld`

java

```
Hello World !
Hello world !
Hello World !
Hello World !
Hello World
```

Méthodes (introduction)

Avertissement : ce cours ne présente qu'une version « édulcorée » des méthodes.
Ne sont abordés que les méthodes statiques et le passage de paramètres.

Méthodes

- Méthodes ⇔ fonctions / procédures
  - *Pour factoriser du code*
  - *Pour structurer le code*
  - *Pour servir de « sous programmes utilitaires » aux autres méthodes de la classe*
  - *…*

- En java plusieurs types de méthodes
  - *Opérations sur les objets (cf. envois de messages)*
  - *Opérations statiques (méthodes de classe)*
    - *exemples*

```
Math.random();
LectureClavier.lireEntier();
```

  - *Pour le moment nous ne nous intéresserons qu'au second type*

Déclaration

- **« Une déclaration de méthode définit du code exécutable qui peut être invoqué, en passant éventuellement un nombre fixé de valeurs comme arguments »** ***The Java Langage Specification*** J. Gosling, B Joy, G. Steel, G. Bracha

- **Déclaration d'une méthode statique**

```
static <typeRetour> nomMethode( <liste de paramètres> ) {
    <corps de la méthode>
}
```

Signature de la méthode

- ***exemple***

```
static double min(double a, double b) {
    if (a < b)
       return a;
    else
       return b;
}
```

Déclaration

- **Déclaration d'une méthode statique**

```
static <typeRetour> nomMethode( <liste de paramètres> ) {
    <corps de la méthode>
}
```

  - **<typeRetour>**
    - Quand la méthode renvoie une valeur (fonction)indique le type de la valeur renvoyée

```
static double min(double a, double b)
static int[] premiers(int n)
```

    - **void** si la méthode ne renvoie pas de valeur (procédure)

```
static void afficher(double[][] m)
```

Déclaration

- **Déclaration d'une méthode statique**

```
static <typeRetour> nomMethode( <liste de paramètres> ) {
    <corps de la méthode>
}
```

  - `<liste de paramètres>`
    - vide si la méthode n'a pas de paramètres

```
static int lireEntier()
static void afficher()
```

    - une suite de couples *type identificateur* séparés par des virgules

```
static double min(double a, double b)
static int min(int[] tab)
```

Déclaration

- **Déclaration d'une méthode statique**

```
static <typeRetour> nomMethode( <liste de paramètres> ) {
    <corps de la méthode>
}
```

  - **<corps de la méthode>**
    - suite de déclarations de variables locales et d'instructions
    - si la méthode à un type de retour le corps de la méthode doit contenir au moins une instruction **return *expression*** où ***expression*** délivre une valeur compatible avec le type de retour déclaré.

```
static double min(double a, double b) {
    double vMin;
    if (a < b)
       vMin = a;
    else
       vMin = b;
    return vMin;
}
```

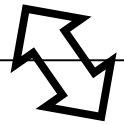
Variable locale

Instruction de retour

Déclaration

- *si la méthode à un type de retour le corps de la méthode doit contenir **au moins** une instruction* ***return expression*** *...*

```
static boolean contient(int[] tab, int val b) {

   boolean trouve = false;
   int i = 0;
   while ((i < tab.length) && (! trouve)) {
           if (tab[i] == val)
              trouve = true;
        i++;
   }
   return trouve;
}
```



- *Possibilité d'avoir plusieurs instructions* ***return***
- *Lorsqu'une instruction return est exécutée retour au programme appelant*
  - Les instructions suivant le `return` dans le corps de la méthode ne sont pas exécutées

```
for (int i = 0; i < tab.length; i++)
   if (tab[i] == val)
     return true;

return false;
```

Déclaration

- **`return`** *sert aussi à sortir d'une méthode sans renvoyer de valeur (méthode ayant* **`void`** *comme type retour)*

```
static void afficherPosition(int[] tab, int val) {

  for (int i = 0; i < tab.length; i++)
     if (tab[i] == val){
       System.out.println("La position de " + val + " est " + i);
       return;
     }

  System.out.println(val + " n'est pas présente dans le tableau");

}
```

Déclaration

- ***<corps de la méthode>***
  - suite de déclarations de variables locales et d'instructions
- *Les variables locales sont des variables déclarées à l'intérieur d'une méthode*
  - elles conservent les données qui sont manipulées par la méthode
  - elles ne sont accessibles que dans le bloc dans lequel elles ont été déclarées, et leur valeur est perdue lorsque la méthode termine son exécution

```
static void method1(...) {
    int i;
    double y;
    int[] tab;
    ...
}
```

Possibilité d'utiliser le même identificateur dans deux méthodes distinctes
pas de conflit, c'est la déclaration locale qui est utilisée dans le corps de la méthode

```
static double method2(...) {
    double x;
    double y;
    double[] tab;
    ...
}
```

Invocation

- **Déclaration**

```
static <typeRetour> nomMethode( <liste de paramètres> ) {
      <corps de la méthode>
}
```

- **Appel**

```
nomMethode(<liste de paramètres effectifs>)
```

  - **<liste de paramètres effectifs>**
    - Liste d'**expressions** séparées par des virgules et dont le nombre et le type correspond (compatible au sens de l'affectation) au nombre et au type des paramètres de la méthode

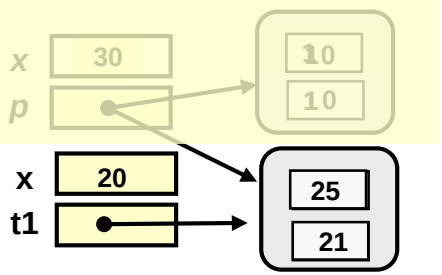
déclaration	appel (invocation)
`static void afficher(){ … }`	`afficher()`
`static double min(double a, double b){ … }`	`min(10.5,x)` avec double x `min(x + y * 3,i)` avec double x,y int i
`static boolean contient(int[] tab,int val){ … }`	`contient(tab1,14)` avec int[] tab1 = new int[10] `contient(tab2,i)` avec int[] tab2 = new int[60] int i

Invocation

- Le passage de paramètres lors de l'appel d'une méthode est un passage par valeur.
  - *À l'exécution le paramètre formel défini dans la signature de la méthode correspond à une variable locale au bloc de la méthode*
  - *Elle est initialisée avec la valeur de l'expression définie par le paramètre effectif.*

```
Class Bidon {

   ...
   static void foo(int x, int[] p) {
      ...
      for (int i = 0; i < p.length; i++)
           p[i] += 10;
      x = x + 10;
      p = new int[2];
      p[0] = 10;
      p[1] = 10;
      ...
   }

}
```

```
int[] t1 = new int[2];
t1[0] = 15;
t1[1] = 11;
int x = 20;
foo(x,t1);
System.out.printl("x " + x);
System.out.println("t1[0] " + t1[0]);
System.out.println("t1[1] " + t1[1]);
```



------> ???

```
x : 20
t1[0] : 25
t1[1] : 21
```

Méthodes statiques

Invocation

- Toute méthode statique d'une classe peuvent être invoquée depuis n'importe quelle autre méthode statique de la classe
  - l'*ordre de déclaration des méthodes n'a pas d'importance*

```
public class A {
   static void m1() {
      ...
   }

   static void m2() {
      m1();
      m3();
   }

   static void m3() {
      ...
   }
}
```

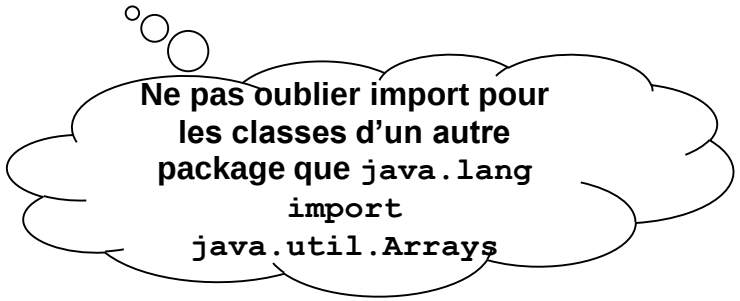
- Pour invoquer méthode d'une autre classe il faut la préfixer par **`NomClasse.`**

```
LectureClavier.lireEntier();
```

```
Math.random();
```

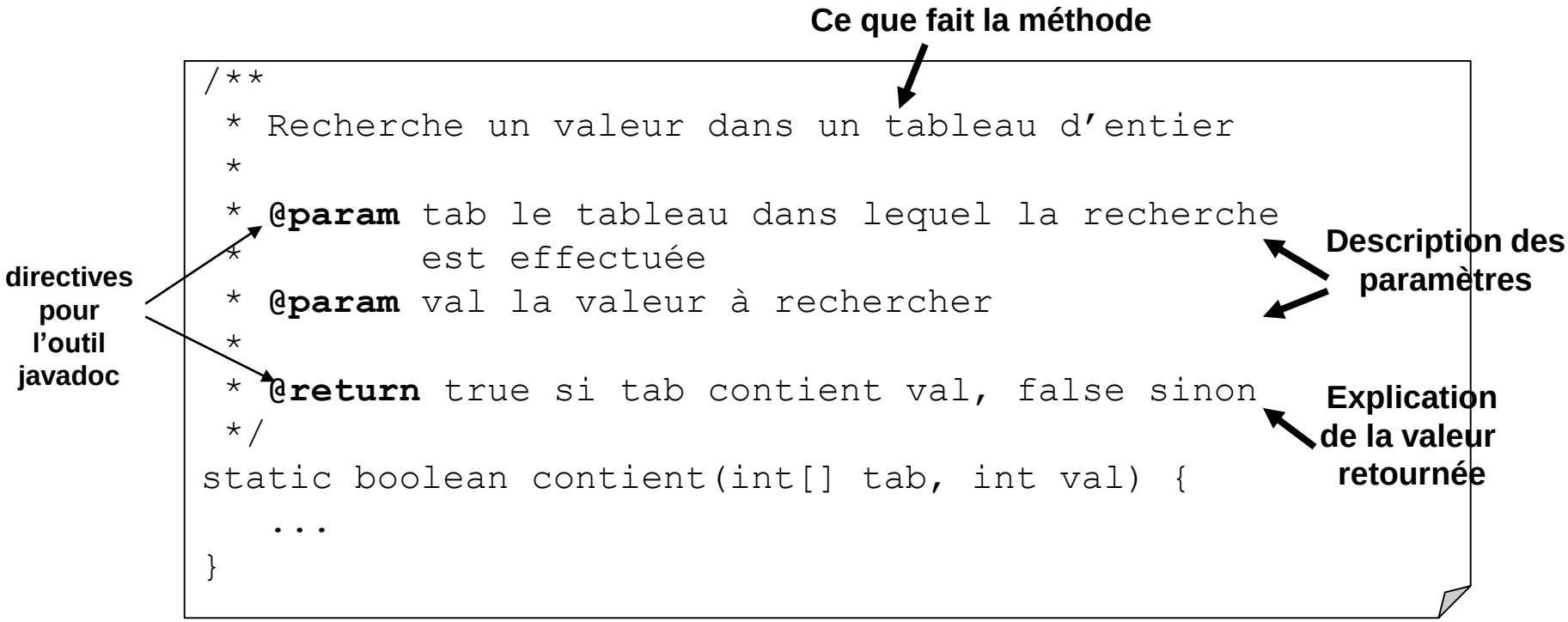
```
Arrays.sort(monTableau);
```

- Possibilité d'avoir des méthodes avec des signatures identiques dans des classes différentes



Commentaires

- Toute déclaration de méthode doit **TOUJOURS** être précédée de son commentaire documentant (exploité par l'outil javadoc)




```
/**
 * Recherche un valeur dans un tableau d'entier
 *
 * @param tab le tableau dans lequel la recherche
 *            est effectuée
 * @param val la valeur à rechercher
 *
 * @return true si tab contient val, false sinon
 */
static boolean contient(int[] tab, int val) {
   ...
}
```

Tableaux en Java (introduction)

Avertissement : ce cours ne présente qu'une version « édulcorée » des tableaux.
Ne sont abordés que les tableaux de types primitifs et tous les aspects « objets » sont masqués.

Tableaux

- Fournissent des collections ordonnées d'éléments
- Composants d'un tableau peuvent être :
  - *des variables des types de base (int, byte, short, long, boolean, double, float, char)*
  - *des références sur des objets (tableaux d'objets)*

*Les tableaux = **objets** en java (et pas seulement une suite d'emplacements mémoire comme en C/C++)*

Déclaration

- **Déclaration**
  - ***typeDesElements[] nomDuTableau;***

  avec
    - *`typeDesElements` un des types de base du langage JAVA (`char, int, float, double...`) ou un nom de classe*
    - *`nomDuTableau` l'identificateur pour désigner le tableau*

    - *la déclaration* ***`typeDesElements nomDuTableau[];`*** *est aussi possible mais des deux formes de déclaration on préférera la première car elle place la déclaration de type en un seul endroit.*

- exemples

```
int vecteurEntiers[];
int[] vecteurEntiers; // identique

Compte[] listeDesComptes;
```

Création

- **`typeDesElements[] nomDuTableau;`**
  - *définit (déclare) un identificateur **`nomDuTableau`** qui permettra de désigner (référencer) un tableau du type déclaré ("tableau d'éléments de type **`typeDesElements`**").*
  - *mais ne crée pas de tableau en mémoire. (**`nomDuTableau == null`**)*

- Pour utiliser un tableau, après la déclaration d'un identificateur permettant de le désigner, il faut ensuite explicitement "créer" ce tableau en mémoire.
  - *la "création" s'effectue à l'aide de l'opérateur **`new`** (utilisé pour créer des objets, or on le verra plus tard les tableaux sont des objets).*
  - ***`nomDuTableau = new typeDesElements[taille]`***

- exemples

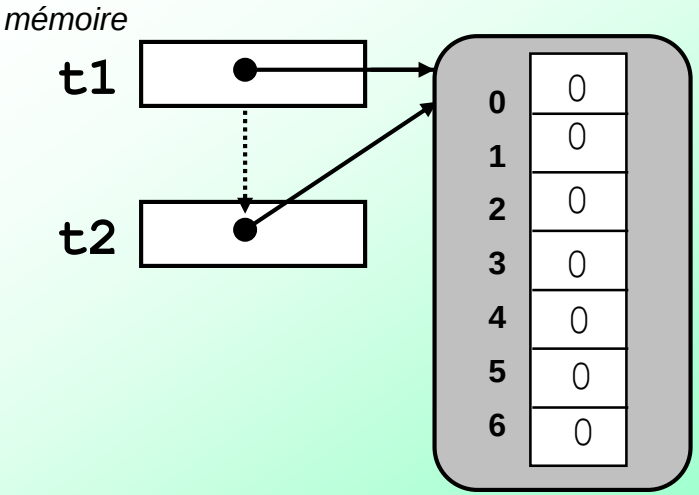
```
int[] vecteurEntiers;
vecteurEntiers = new int[50];

Compte[] listeDesComptes = new Compte[1000];
```

  - *La taille donné à la création est fixe, elle ne peut être modifiée par la suite.*

Création

- La création d'un tableau par **`new`**
  - *alloue la mémoire nécessaire en fonction*
    - *du type du tableau*
    - *de la taille spécifiée*
  - *initialise le contenu du tableau*
    - *type simple : 0*
    - *type complexe (classe) :* ***`null`***

```
int[] t1;
t1 = new int[7];
int[] t2 = t1;
```



Accès aux éléments

- accès à un élément d'un tableau s'effectue à l'aide d'une expression de la forme :

```
nomDuTableau[expression1]
```

  - ***expression1*** *est une expression entière qui définit l'index dans le tableau de l'élément considéré*
  - *comme en C/C++ les éléments d'un tableau sont indexés de* ***0*** *à* ***taille-1***
  - *Java vérifie automatiquement l'indice lors de l'accès (comparaison avec la borne)*
    - *Si hors limites :* `ArrayIndexOutOfBoundsException`
    - *Evite des bugs !*

Accès aux éléments

- **`nomDuTableau.length`** donne la taille du tableau **`nomDuTableau`**
  - *`int[] tabEntiers = new int[10]`*
    `tabEntiers.length` → *`10`* *taille du tableau*
    `tabEntiers.length -1` → *indice max de `tabEntiers`*

- **L'argument `String[] args` du `main` est un tableau de chaînes de caractères `(String)` correspondant aux arguments de la ligne de commande.**

```
public class TestArgs {
 public static void main(String[] args) {
   System.out.println("nombre d 'arguments : " + args.length);
   for (int i =0; i < args.length; i++)
        System.out.println(" argument " + i + " = " + args[i]);
  }
}
```

- **Affiche sur la console les arguments**

```
Java TestArgs toto 23 titi
```

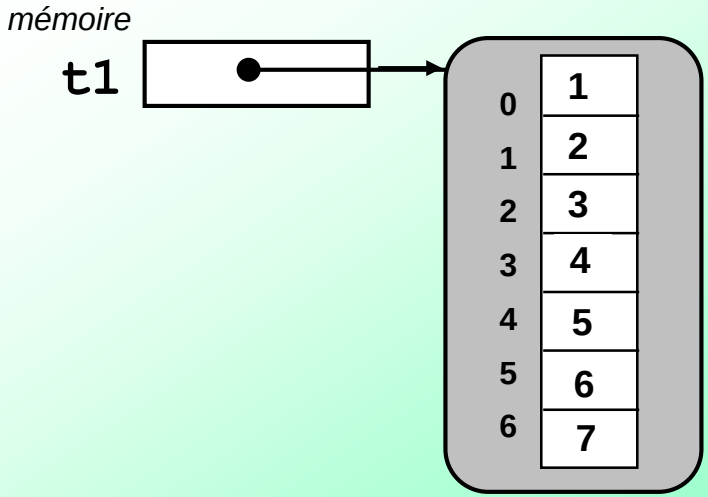
→

```
argument[0] = toto
argument[0] = 23
argument[0] = titi
```

Accès aux éléments

`nomTab[expr entière]` désigne la variable correspondant à l'élément du tableau dont l'index est donné par `expr entière`

```
int[] t1;
t1 = new int[7];

t1[0] = 1;
for (int i=1; i<7; i++)
  t1[i] = t1[i-1]+1;
```



en partie gauche d'une affectation désigne un élément du tableau

dans une expression délivre le contenu (la valeur de cet élément)

Création

- **Une autre manière de créer des tableaux :**
  - *en donnant explicitement la liste de ses éléments à la déclaration (liste de valeurs entre accolades)*
  - *exemples :*

```
int[] t1 = { 1, 2 ,3, 4, 5};

char[] codes = { 'A', 'a', 'B', 'z' };
```

  - *l 'allocation mémoire (équivalent de l'utilisation de* ***new****) est prise en charge par le compilateur*

Tableaux multidimensionnels

- tableau dont les éléments sont eux mêmes des tableaux
- un tableau à deux dimensions se déclarera ainsi de la manière suivante :

```
typeDesElements[][] nomduTableau;
```

  - *exemples*
    - `double[][] matrice;`
    - `Voxel[][][] cubeVoxels;`

Création

- **dimensions du tableau**
  - *ne sont pas spécifiées à la déclaration (comme pour les tableaux à une seule dimension).*
  - *indiquées que lors de la création*
    - *obligatoire que pour la première dimension.*
    - *autres dimensions peuvent n'être spécifiées que lors de la création effective des tableaux correspondants.*

```
double [][] matrice = new double[4][4];
```

Création d 'une matrice 4x4 de réels

```
double [][] matrice = new double[4][];
for (int i=0; i < 4; i++)
   matrice[i] = new double[4];
```

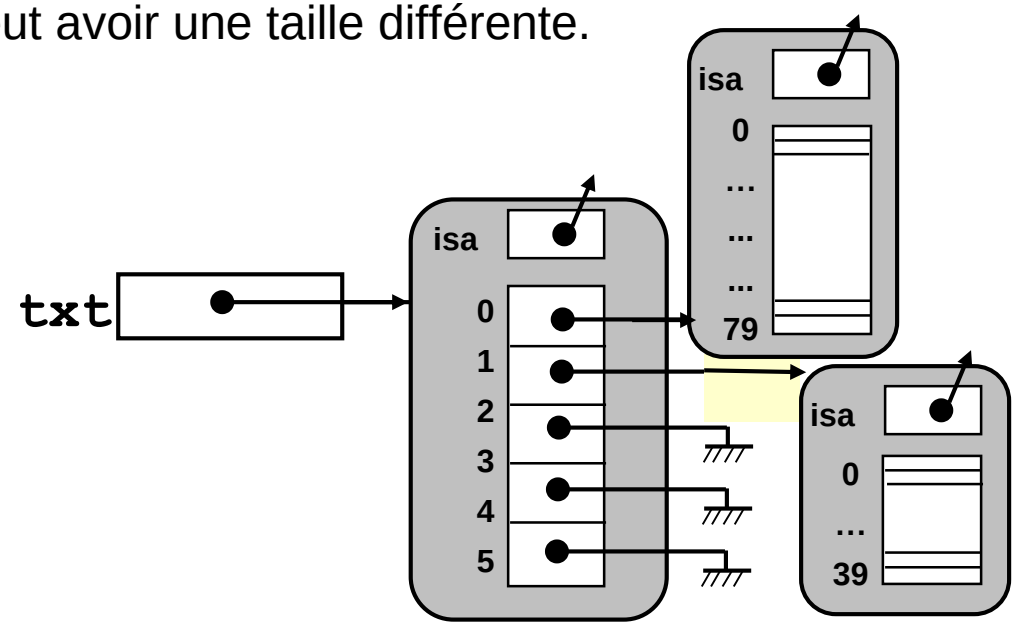
Les 3 écritures sont équivalentes

```
double [][] matrice;
matrice = new double[4][];
for (int i=0; i < 4; i++)
   matrice[i] = new double[4];
```

Tableaux

- chaque tableau imbriqué peut avoir une taille différente.

```
char [][] txt;
txt = new char[6][];
txt[0] = new char[80];
txt[1] = new char[40];
txt[2] = new char[70];
...
```



Accès aux éléments

- **accès aux éléments d'un tableau multidimensionnel (exemple 2d)**
  - *accès à un élément d'un tableau s'effectue à l'aide d'une expression de la forme :*

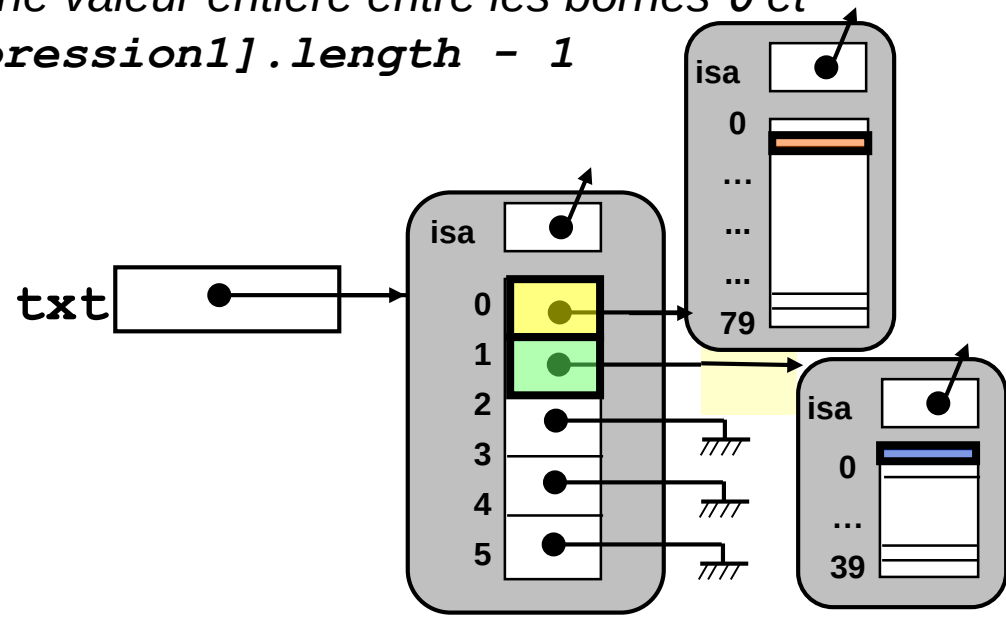
    nomDuTableau[expression1][expression2]

    où

    – *expression1 délivre une valeur entière entre les bornes* **0** *et* ***nomDuTableau.length - 1***

    – *expression2 délivre une valeur entière entre les bornes* **0** *et* ***nomDuTableau[expression1].length - 1***

```
char[][] txt = new char[5][];
txt[0] = new char[80];
txt[1] = new char[40];
txt[0][1] = 'S';
txt[1][0] = 'H';
```



Création

- **Comme pour tableaux unidimensionnels possible de créer un tableau multidimensionnel en donnant explicitement la liste de ses éléments à la déclaration (liste de valeurs entre accolades)**
  - *exemples :*

```
int[][] t1 = {
        { 1, 2, 3, 4, 5},
        { 6, 7, 8, 9, 10},
    };
```

```
int[][] t1 = new int[2][5];
t1[0][0] = 1; t1[0][1] = 2; ...
t1[1][0] = 6; t1[1][1] = 7;...
```

Cette virgule finale n'est pas une faute de frappe, elle est optionnelle et est juste là pour permettre une maintenance plus facile de longues listes (et faciliter les couper/ coller des programmeurs pressés… :-)

Tableaux

à propos de la classe Arrays

- package **`java.util`** définit une classe, **`Arrays`**, qui propose des méthodes statiques (de classe) pour le tri et la recherche dans des tableaux.

Exemple : tri d'un tableau

```
// tableau de 1000 réels tirés au hasard
// dans l 'intervalle [0..1000[
double[] vec = new double[1000];
for (int i = 0; i < vec.length; i++)
      vec[i] = Math.random()*1000;

// tri du tableau
Arrays.sort(vec);

// affiche le tableau trié
for (int i = 0; i < vec.length; i++)
      System.out.print(vec[i] + " " );
```

Le tri.
Dans l 'implémentation de SUN une variation du QuickSort
(O(n*log(n))

Tableaux

à propos de la classe Arrays

Exemple : recherche dans un tableau

```
// tableau de 1000 entiers tirés au hasard
// dans l 'intervalle [0..1000[
int[] vec = new int[1000];
for (int i = 0; i < vec.length; i++)
       vec[i] = (int) (Math.random()*1000);

 // tri du tableau
 Arrays.sort(vec);

 // recherche de la valeur 500
 int pos = Arrays.binarySearch(vec,500);

// utilisation des résultats de la recherche
if (pos >= 0)
  System.out.println("position de 500 : " + pos);
else {
  System.out.println("500 n 'est pas dans le tableau");
  System.out.println("position d 'insertion : " + (-(pos+1)));
}
```

Il faut que le tableau soit trié avant toute recherche

La recherche

Tableaux

à propos de la classe Arrays

- **pour chaque type de tableau**
  - *Des méthodes de recherche*
    - *int binarySearch(char[ ] a) , int binarySearch(int[ ] a) ….*
      … int binarySearch(Object[ ] a)
  - *Des méthodes de tris*
    - *sort(char[ ] a) , sort(int[ ] a) ….. sort(Object[ ] a)*
    - *sort(char[ ] a, int fromIndex, int toIndex) , ...*
  - *Des méthodes pour remplissage avec une valeur*
    - *fill(char[ ] a, char val) , fill(int[ ] a, long val) …..*
    - *fill(char[ ] a, char val, int fromIndex, int toIndex) , …*
  - *Des méthodes de test d'égalité*
    - *boolean equals(char[ ] a1, char[ ] a2), boolean equals(int[ ] a1, int[ ] a2), …..*

Tableaux

à propos de java.util

- Les tableaux sont des structures de données élémentaires
- Le package **`java.util`** contient plein de classes « sympa » pour la gestion de structures de données plus évoluées (collections) :
  - *listes*
  - *ensembles*
  - *arbres*

- mais pour apprécier il faudra être un peu patients…
  parlons d ’abord d’objets,
  d’héritage,
  de classes abstraites
  et d ’interfaces ;-) !