

Tableaux d'objets

- Des tableaux contenant des (références à des) objets d'une classe donnée peuvent être déclarés et créés ***pour toute classe (prédéfinie*** ou bien écrite par le programmeur).
- Par exemple, un tableau de chaînes de caractères (classe String) se déclare et se dimensionne ainsi :

```
String[] tabChaines;
```

```
tabChaines = new String[2];
```

- ATTENTION : le dimensionnement d'un tableau d'objets crée uniquement des « cases » pour stocker des ***références aux*** objets, mais *pas les objets eux-mêmes*. Ces références valent initialement null, et il faut donc les faire ensuite pointer vers les objets voulus par des affectations.

Par exemple, une suite possible des instructions précédentes est :

- `tabChaines[0] = "OUI";`
- `tabChaines[1] = "NON";`

Références constantes

- Ce sont des références associées de façon définitive à un objet ou tableau donné
- mot-clé final :

```
final double[] tab;  
tab = new double[10];  
//La référence tab est figée :  
tab = new double[20]; //ici ERREUR  
// Le tableau reste modifiable :  
tab[0] = 1.732; // OK  
tab[0] = 3.; // OK
```

Méthodes

- Méthodes \leftrightarrow 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).

Attribut et méthode statique

- On rappelle qu'une méthode ou un attribut auxquels n'est pas appliqué le modificateur **static** sont dits d'instance.
- Un attribut ou une méthode statique (i.e. déclaré avec le mot **static**) est aussi dit **attribut de classe** ou **méthode de classe**.
- Un attribut déclaré **static** (i.e. statique ou de classe) existe dès que sa classe est chargée en mémoire, en dehors et indépendamment de toute instanciation.

Attribut et méthode statique

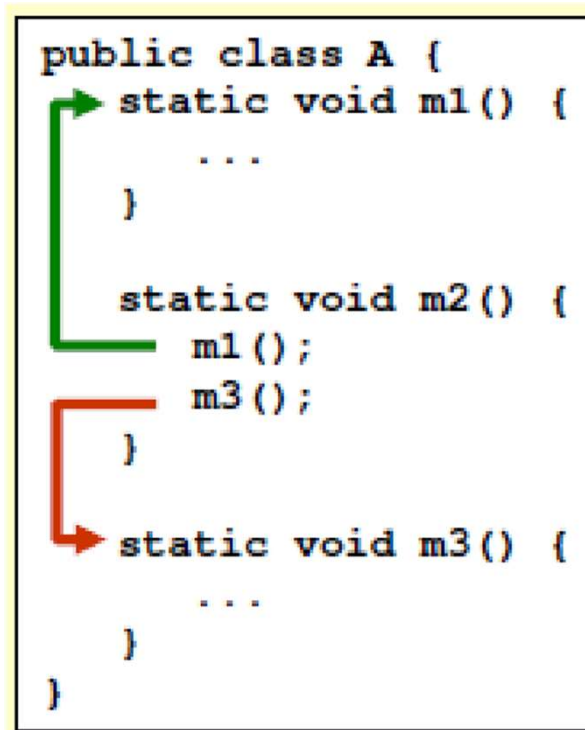
- Quelque soit le nombre d'instanciation de la classe (0, 1, ou plus) un attribut de classe, existe en un et un seul exemplaire.
- Un tel attribut sera utilisé un peu comme une variable globale d'un programme non objet.
- Une méthode, pour être de classe, ne doit pas manipuler, directement ou indirectement, d'attributs non statiques de sa classe.
- De l'extérieur de sa classe ou d'une classe héritée, un attribut ou une méthode de classe pourront être utilisés précédés du nom de sa classe :

`nom_d'une_classe.nom_de_l'attribut_ou_méthode_de_classe`

<liste de paramètres>

- vide si la méthode n'a pas de paramètres : `static int lireEntier()`
ou `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)`
- Toute méthode statique d'une classe peut être invoquée depuis n'importe quelle autre méthode statique de la classe
➔ *L'ordre de déclaration des méthodes n'a pas d'importance*

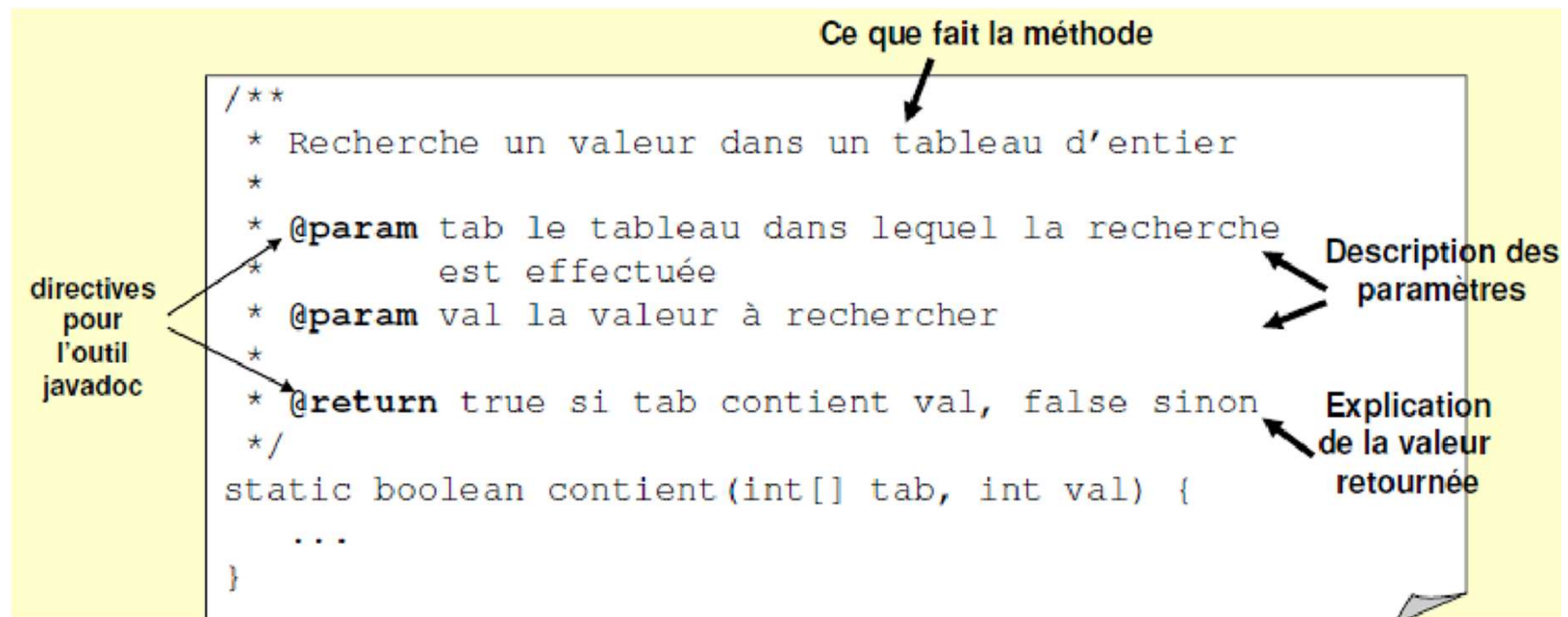
```
public class A {  
    static void m1() {  
        ...  
    }  
  
    static void m2() {  
        m1();  
        m3();  
    }  
  
    static void m3() {  
        ...  
    }  
}
```

A diagram illustrating method calls within a class. A green arrow originates from the 'm1()' call inside the 'm2()' method and points back to the 'm1()' method definition. A red arrow originates from the 'm3()' call inside the 'm2()' method and points to the 'm3()' method definition.

- Pour invoquer méthode d'une autre classe il faut la préfixer par **NomClasse**. **Exemple** : `Math.random()`;
- Possibilité d'avoir des méthodes avec des signatures identiques dans des classes différentes

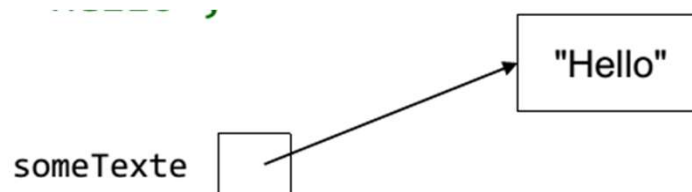
Commentaire des fonctions

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



Les chaînes de caractères

- En *Java* les chaînes de caractères sont représentées par des objets de type **String** (une classe prédéfinie).
- Les chaînes de caractères ne font donc pas partie des types primitifs mais sont des types référence. Il s'agit d'une classe définie dans l'API Java (Dans le package `java.lang`).
- Une syntaxe particulière est utilisée pour définir des littéraux de type String : on entoure le texte avec des guillemets ("...").
- On peut insérer des séquences d'échappement (identiques à celles définies pour le type **char**) dans les littéraux de type String.
- La déclaration et création d'une variable de type String s'effectue de la manière suivante : **String someText = "Hello";**



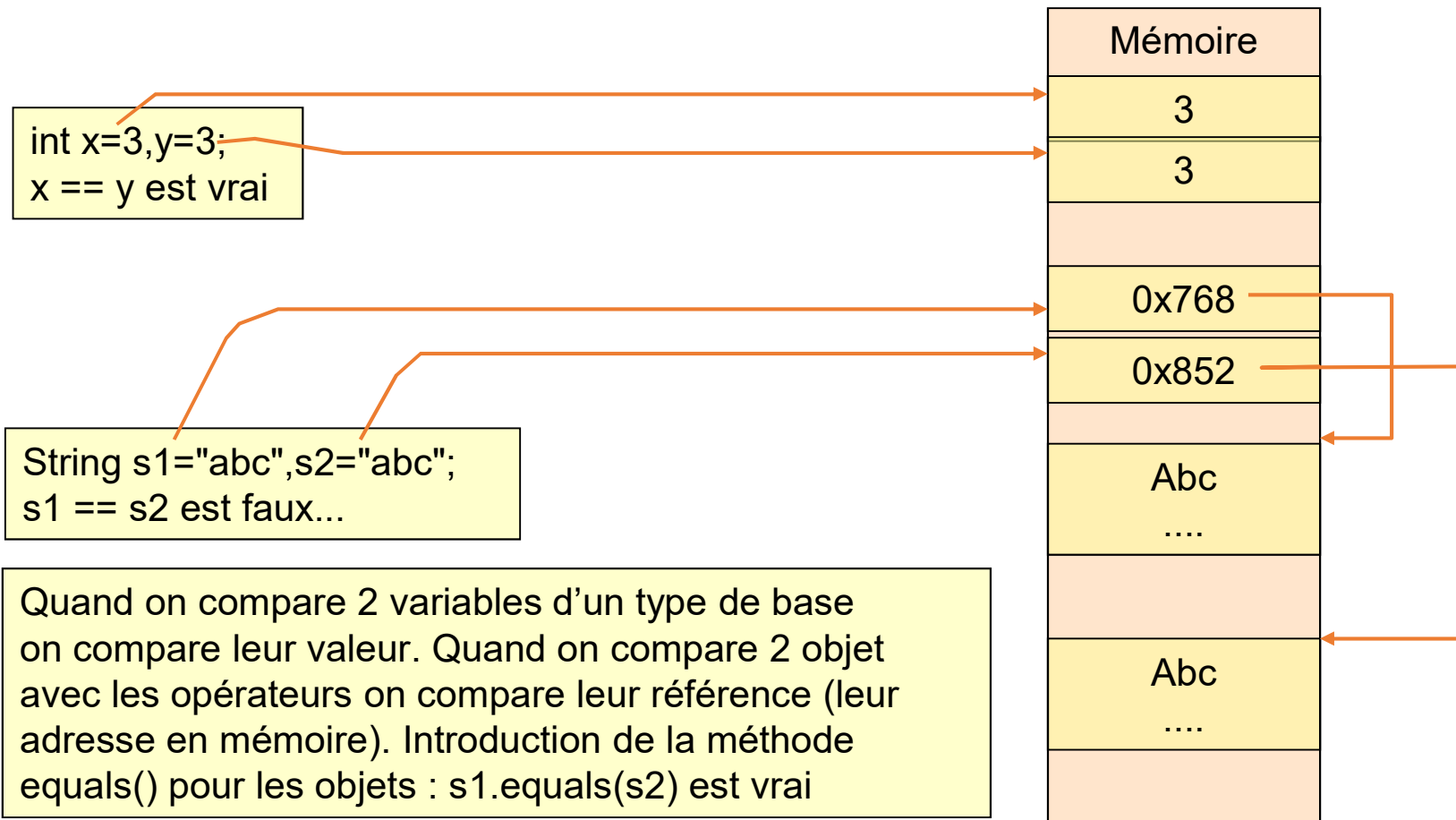
Comparaison et concaténation de chaînes

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

- Comme pour tous les types référence, l'opérateur "==" **compare les références** et non pas les objets référencés (les chaînes de caractères).
- La méthode **equals()** permet, elle, de **comparer deux chaînes de caractères** (objets référencés) et retourne true si elles sont égales.
- 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 ? " */
```

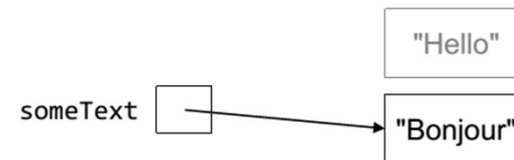
Différences entre objets et types de base



Objets String immuables

- Les objets de type String sont **immuables** : une fois créés ils ne peuvent plus être modifiés. Naturellement, la variable de type référence peut changer de valeur, et pointer vers une nouvelle chaîne (créée à un autre emplacement).

```
String someText = "Hello";  
someText = "Bonjour";
```



- Si l'on manipule fréquemment des objets de type String (à l'intérieur de boucles par exemple) cela peut conduire à la création d'un nombre considérable d'objets (temporaires) avec un coût non négligeable (ressources mémoires et temps d'exécution).
- Il est préférable dans ce cas de déclarer et utiliser des objets de type StringBuffer ou StringBuilder qui sont eux modifiables.

Opérations sur les chaînes

- La classe `String` dispose d'un grand nombre de méthodes pour traiter les chaînes de caractères. Les exemples qui suivent n'en sont qu'une illustration sommaire.
- La méthode **`length()`** permet de connaître la longueur d'une chaîne de caractères (nombre de caractères).
- La méthode **`charAt()`** permet de retrouver le caractère situé à la position n d'une chaîne ($0 \leq n \leq \text{length}() - 1$).
- La méthode **`substring()`** retourne une sous-chaîne d'une chaîne donnée. Les paramètres déterminent la position initiale et la position finale (+1) de la sous-chaîne dans la chaîne originelle.

```
String name = "James Bond";  
int    i = name.length();           // i = 10  
char   c = name.charAt(1);          // c = 'a'  
String s = name.substring(2, 4);    // s = "me"  
String t = "Yes".toUpperCase();    // t = "YES"
```

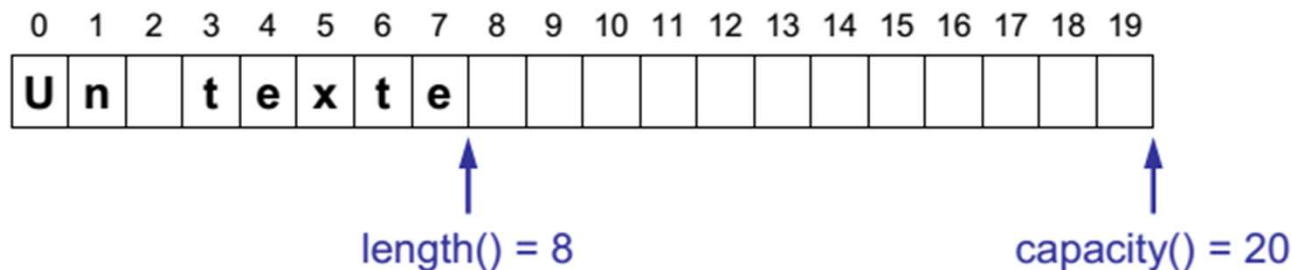
Opérations sur les chaînes

- La méthode **split()** permet de découper une chaîne de caractères sur la base d'un séparateur qui est exprimé sous la forme d'une *expression régulière (regex)*. La méthode retourne le résultat dans un tableau de String qui contient les fragments découpés.
- La syntaxe à utiliser pour les expressions régulières est définie dans la classe Pattern (java.util.regex).

```
public static void main(String[] args) {  
    String path = "usr/lib/ruby/test.rb";  
    String[] elems;  
  
    String sepPattern1 = "/"; // Slash separator  
    elems = path.split(sepPattern1); elems → { "usr", "lib", "ruby", "test.rb" }  
  
    String sepPattern2 = "/|\\."; // Slash or dot separator  
    elems = path.split(sepPattern2); elems → { "usr", "lib", "ruby", "test", "rb" }  
}
```

Classe StringBuffer

- Contrairement à la classe String, la classe StringBuffer permet de créer des chaînes de caractères modifiables (la taille et le contenu peuvent varier durant l'exécution de l'application).
- La notion de **capacité** (*Capacity*) représente la taille du *buffer* interne qui mémorise la chaîne de caractères (nombre total de caractères disponibles avant de devoir agrandir la taille du buffer interne).
- La capacité est automatiquement augmentée lorsque c'est nécessaire. La capacité initiale peut être définie lors de la création d'un objet StringBuffer (paramètre du constructeur, 16 par défaut).
- Ne pas confondre capacité et longueur d'un objet StringBuffer.



Classe StringBuffer



- Pour **créer un objet** de type StringBuffer, on doit utiliser l'opérateur new(...) (pas de syntaxe simplifiée).

```
StringBuffer someText = new StringBuffer("Hello");
```

- Création en définissant une capacité initiale :

```
// Création d'une chaîne vide avec une capacité de 2000 caractères
StringBuffer buf= new StringBuffer(2000);
// Ajout d'une chaîne littérale (String)
buf.append("Bonjour"); // Utilisation
System.out.println(buf.length()); // 7
System.out.println(buf.capacity()); // 2000
char c=buf.charAt(0);
buf.setCharAt(0,'L');
buf.insert(3,"gues ");
buf.append("née");
buf.deleteCharAt(6);
String s=buf.toString();
String s2=buf.substring(7,11);
// ATTENTION : pour StringBuffer,
// equals() ne teste pas l'égalité des chaînes contenues
// pas de + entre StringBuffer (par contre buf+s OK si s String, mais produit une String)
```