

# Java

## Les bases

# Les types de base

# Les types de données

- Il existe deux types de données en Java
  - Les types primitifs
  - Les objets
- En Java, on ne manipule que des références sur les objets
  - Déclarer un objet en Java revient à déclarer un pointeur en C
  - Les objets sont systématiquement passés par référence dans les méthodes
  - La référence **null** est l'équivalent du pointeur nul en C/C++ (ne se réfère à rien)

# Initialisation des variables

- Les variables sont automatiquement initialisées à :
  - 0 / false pour les types primitifs
  - null pour les objets
- **⚠ MAIS** : Il n'est pas possible d'utiliser une variable sans l'avoir explicitement initialisée (vérifié à la compilation)

```
int i; // vaut 0
i++; // erreur de compilation
```

```
String s; // vaut null
s = s.toUpperCase (); // erreur de compilation
```

# Vérification des types

- La vérification des types est très stricte en Java.
- A part quelques rares exceptions (autoboxing, toString (), ...) il n'y a pas de conversion implicite en Java

```
int i = 1;
boolean b = i; // KO
double d = 3.14;
float f = d; // KO
```

```
int i = 1;
boolean b = (i == 1); //OK
double d = 3.14;
float f = ( float )d; // OK
```

- Les nombres à virgule écrits « en dur » sont implicitement des double. Il faudra les caster explicitement lors de l'initialisation d'un float

```
float f = (float) 3.14;
float f2 = 3.14 f ;
```

# Les types primitifs

- Ce sont les seuls types qui ne sont pas des objets en Java
- On retrouve les types de C/C++ :
  - boolean (vaut true / false)
  - char
  - byte
  - short, int , long
  - float , double

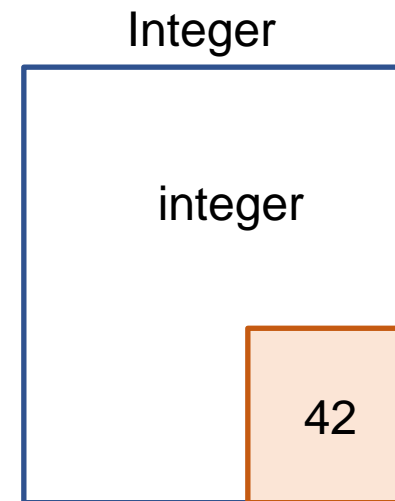
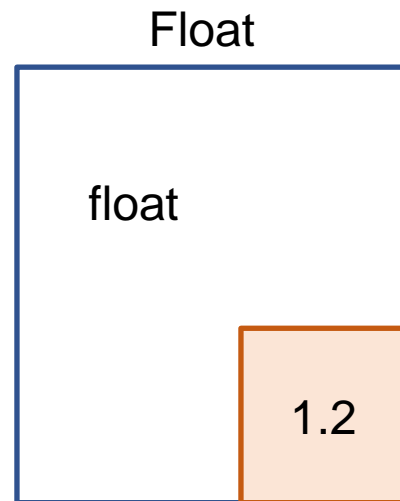
# Les classes enveloppantes

- A chaque type primitif est associé une classe.

Type primitif	Classe enveloppante
boolean	Boolean
char	Character
byte	Byte
short	Short
integer	Integer
long	Long
float	Float
double	Double

# Les classes enveloppantes

- A chaque type primitif est associé une classe.
- Ces classes, appelées classes enveloppantes (***wrapper classes***), encapsulent des types primitifs associés.
- Exemples :





# Intérêt des classes enveloppantes

- Comme toute fonction doit être une méthode de classe, les fonctions utilitaires permettant de manipuler chaque type se trouvent dans les classes correspondantes.
  - Exemple : conversion chaîne "42" en entier

C/C++

```
atoi("42");
```

Java

```
Integer.parseInt("42");
```

# Intérêt des classes enveloppantes

- Comme toute fonction doit être une méthode de classe, les fonctions utilitaires permettant de manipuler chaque type se trouvent dans les classes correspondantes.
- Ces classes permettent d'utiliser du code qui manipule des objets.
  - On peut appeler une méthode qui prend un objet en paramètre avec un type primitif, qui est alors automatiquement « emballé ». Ce mécanisme s'appelle ***autoboxing***.

# Les chaînes de caractères

# Les chaînes de caractères

- En Java, les chaînes de caractères sont modélisées par la classe **String**
- La méthode **length** renvoie la longueur de la chaîne
- L'opérateur **+** permet de concaténer des chaînes, et d'y inclure les valeurs de types primitifs (conversion implicite)

```
String s1 = "Hello, ";
int l = s1.length();
length();// vaut 7
String s2 = "World !";
String s3 = s1 + s2; // vaut "Hello, World
String s4 = "Length : " + l; // vaut "Length
```

# Opérations de base sur les chaînes

- Les méthodes de la classe String permettent d'effectuer quelques opérations courantes sur les chaînes.

Méthode	Effet
indexOf(String s)	Renvoie l'indice de la première occurrence de s
replaceAll (String old , String new)	Remplace toutes les occurrences de old par new
split(String s)	Coupe la chaîne selon le délimiteur s
trim()	Retire les espaces en début et fin de chaîne
...	

- Quand vous en aurez besoin, consultez la page de [String](#) de l'API Java pour consulter les méthodes disponibles.

# Documentation de l'API Java (« JavaDoc »)



Dès qu'un besoin vous paraît basique/classique, ayez le réflexe de consulter l'API Java [en ligne](#) : il y a certainement une classe qui y répond !

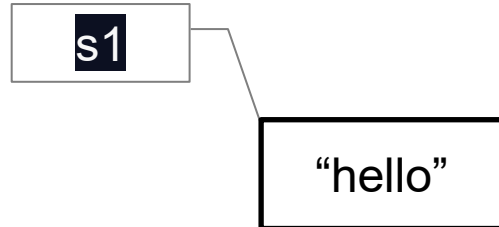


Cette documentation est créée par [javadoc](#), un outil qui génère automatiquement des pages HTML à partir de commentaires spéciaux, contenant des « tags ». Vous pouvez également l'utiliser dans vos projets ! Plus d'infos [ici](#).

# Les chaînes de caractères sont immuables

- Toute tentative de modification sur une chaîne crée une nouvelle chaîne.

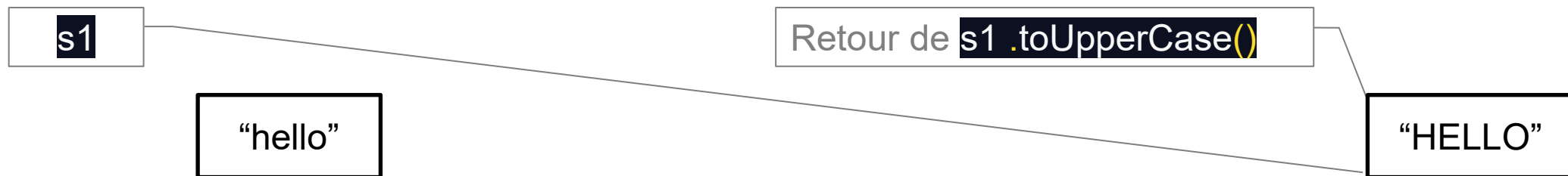
```
String s1 = "Hello";  
s1.toUpperCase();
```



# Les chaînes de caractères sont immuables


- Toute tentative de modification sur une chaîne crée une nouvelle chaîne.

```
String s1 = "Hello";  
s1 = s1.toUpperCase();
```





# Comparaison de chaînes

-  Comme en C, l'opérateur `==` compare l'adresse des chaînes en mémoire, et non leur contenu.
- Pour tester si deux chaînes sont égales, il faut utiliser la méthode `equals()`

```
String s1 = "A";
String s2 = "a";
String s3 = s2.toUpperCase(); // vaut "A"
boolean b1 = (s1 == s3); // vaut false
boolean b2 = s1.equals(s3); // vaut true
```

# Conversions nombre / chaîne

- Pour convertir un nombre en chaîne, utiliser la méthode `toString()`. des classes enveloppantes :
  - Exemple : int vers String

```
int i = 42;
String s = Integer.toString(i); // vaut "42"
```

- Pour convertir une chaîne en nombre, utiliser la méthode `parseXXX()` des classes enveloppantes :
  - Exemple : String vers float

```
String s = "3.14";
float f = Float.parseFloat (s); // vaut 3.14
```

# Les tableaux

# Les tableaux

- En Java, les tableaux sont des objets : ils sont donc manipulés par référence
- Un tableau possède un attribut `length` qui contient sa taille
- `t[i]` permet d'accéder à la case d'indice  $i$  du tableau  $t$
- Les indices commencent à 0

# Les tableaux

- Initialisation d'un tableau vide :

```
type[] array = new type[taille];
```

- Les règles d'initialisation par défaut s'appliquent sur chaque case

- Initialisation d'un tableau avec des valeurs :

```
type[] array = {valeur1, valeur2, ...};
```

- Exemples :

```
float[] a1 = new float[2]; // contient [0.0,  
String[] a2 = new String[ 3]; // contient [null, null, null]  
int[] a3 = {1,2,3,4}; // contient [1, 2, 3,
```

# Parcours de tableaux

- On peut parcourir un tableau comme on le ferait en C/C++ :

```
int[] array = {1,2,3,4};
for(int i =0; i < array.length ; i++){
    int value = array[i];
    System.out.println(i);
}
```



```
1
2
3
4
```

# Parcours de tableaux

- Java possède une boucle de type « for each » qui évite d'explicitement manipuler des indices :

```
int[] array = {1,2,3,4};
for(int value : array){
    System.out.println(value);
}
```



```
1
2
3
4
```

- Ici, la variable value va prendre successivement la valeur de chaque case du tableau

# Les tableaux multidimensionnels

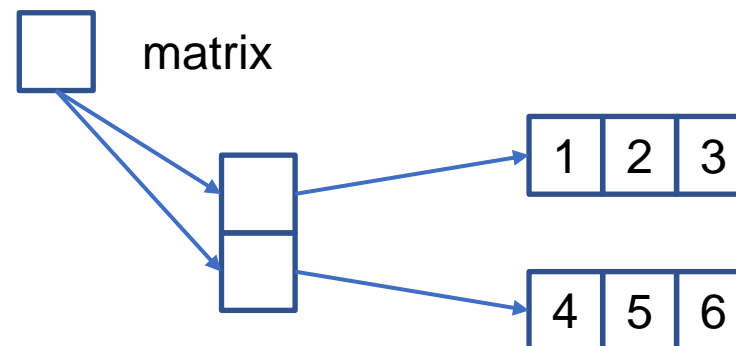
- Initialisation d'un tableau vide à N dimensions:

```
type[]...[] array = new type[dim1][dim2]...[dimN]
```

- Les règles d'initialisation par défaut s'appliquent sur chaque case

- Exemple d'initialisation avec des valeurs :

```
int[][] matrix = {  
    {1, 2, 3},  
    {4, 5, 6}  
};
```





# Les entrées / sorties

# La classe System

- La classe System permet d'interagir avec le système d'exploitation
- Elle permet notamment de manipuler :
  - Les variables d'environnement
  - Les flux d'entrée / sortie

# Les flux d'entrée / sortie

- **System.out** modélise la sortie standard
  - La méthode **print()** permet d'afficher des données avec un formatage automatique.
  - La méthode **println()** ajoute un retour à la ligne (portable)
- **System.in** modélise l'entrée standard
  - La méthode **read()** permet de lire des octets
  - Elle est fastidieuse à utiliser car il faut manipuler les octets et faire les conversions à la main

# La classe Scanner

- La classe Scanner fait partie du package `java.util`
- Elle découpe la chaîne en morceaux selon un délimiteur (espace par défaut, modifiable)
- On peut convertir chaque morceau dans un type cible. Une exception est levée si la conversion échoue.

Méthode	Renvoie le prochain morceau sous forme de...
<code>next()</code>	String
<code>nextInt()</code>	int
<code>nextFloat()</code>	float
...	

- Ne pas oublier de fermer le Scanner après utilisation, avec la méthode `close()`

# Hello, {Votre nom ici} !

```
public class HelloYou {  
    public static void main(String[] args) {  
        Scanner s = new Scanner(System.in); // création du Scanner  
        String name = s.next (); // lecture d'une chaîne bloquant  
        System.out.println ("Hello, " + name + "!");  
        s.close(); // fermeture du Scanner  
    }  
}
```