

# Programmation en Java : Envoi n°1 - Partie 2

---

05 janvier 2026

**String**

# String

En Java, les chaînes de caractères n'appartiennent pas aux types primitifs, mais sont des objets de la classe *String*, les premiers et principaux que nous manipulerons.

**Les objets se distinguent par :**

- **leur type** commence par une majuscule
- **leur initialisation** requiert le mot-clef *new* suivi du nom de la classe
- **leur stockage** en mémoire est particulier et induit que :
  - leurs comparaisons nécessitent des précautions
  - leur manipulation se fait par les fonctions de la classes, apellées *méthodes*.

```
1 String firstName = new String("Nicolas");  
2 firstName == "Nicolas" // Retourne false  
3 firstName.equals("Nicolas") // Retourne true
```

# String

Cependant, leur utilisation systématique a conduit à développer quelques raccourcis :

```
1 String lastName = new String("Terrapon");  
2 // peut etre simplifiee en  
3 String lastName = "Terrapon";  
4  
5 String name = firstName.concat(" ").concat(lastName);  
6 // devient  
7 String name = firstName + " " + lastName
```

La syntaxe des méthodes est donc variable.methode(parametresOptionnels)

**Pour connaître l'ensemble des méthodes d'une classe**, rendez-vous sur :

<https://docs.oracle.com/en/java/javase/25/docs/api/index.html>

# String

Parmi les méthodes les plus fréquemment utilisés, on trouve :

```
1 String b = "Bonjour";
2 c = b.contains("jour"); // Equivalent du Python : "jour" in s
3 b.replace('o', 'i'); // "Binjiur"
4 b.substring(2,5); // "njo", from (inclus) - to (exclus)
5 b.toUpperCase(); // "BONJOUR"
6 for(int i=0 ; i<b.length(); i++) { // Longueur de la chaîne
7     System.out.println(i + " " + b.charAt(i));
8     // Tout est converti en String grace au " "
9 } // Affiche "0 B", "1 o", ..., "6 r"
```

Ces méthodes ne modifient pas la valeur/chaîne stockée dans la variable *b* : on parle de *type immuable* pour le type String. Pour conserver ces modifications/informations, il faut enregistrer le résultat dans une variable !

# Classes Enveloppes

# Classes Enveloppes

Pour chaque type primitif, il existe également une classe enveloppe. Ces classes sont notamment utilisées pour la conversion depuis un String :

```
1 String s = "3";  
2 int c = Integer.parseInt(s); // 3  
3 double d = Double.parseDouble(s); // 3.0  
4 boolean e = Boolean.parseBoolean(s); // false
```

On remarque une **différence de syntaxe** : ces méthodes ne s'appliquent pas à l'objet mais à la classe et apparaissent d'ailleurs avec le mot-clé *static* dans la documentation. Les classes enveloppes permettent également d'**interroger le type**, ainsi que certaines de ses **propriétés/attributs** :

```
1 c.getClass(); // Renvoie Integer  
2 Integer.BYTES; // 32  
3 Integer.MAX_VALUE // 2 147 483 647
```

## **Autres Librairies**

## Autres Librairies

De nombreuses autres librairies peuvent s'avérer utiles. Par exemple, la librairie MATH

```
1 Math.PI; // Valeur plus pr cise qu'a la main
2 Math.sqrt(64); // 8
3 Math.pow(2, 8); // 256.0
4 Math.round(4.6); // 5
5 int randNb = (int)(Math.random() * 101); // 0 to 100
```

Et vous utilisez la librairie *System* depuis votre 1er programme sans vous en rendre compte :

```
1 System.out.println("Hello World !");
```

## Autres Librairies

Alors que certaines classes/librairies (comme celles précédemment présentées) sont chargées au démarrage de Java, **d'autres nécessitent une importation** de notre part, comme *Scanner* :

```
1 import java.util.Scanner; // Au tout debut du fichier
2
3 // Et dans une fonction
4 Scanner sc = new Scanner(System.in);
5 int i = sc.nextInt(); // ou double j = sc.nextDouble();
6 String p = sc.nextLine();
```

*Scanner* permet de récupérer l'*input* clavier de l'utilisateur.

## Autres Librairies

À l'issue du TDP1.1, vous avez une première classe que vous allez être amenés à appeler depuis un autre programme, dans une autre classe.

Si ces deux programmes se situent dans le même dossier, pas besoin d'import, sinon il faudra intégrer les lignes suivantes :

- dans la classe contenant les fonctions

```
1 package folderName.subFolder; // A remplacer par le nom ou  
   chemin jusqu'au fichier, avec des "." pour séparateurs
```

- dans la nouvelle classe souhaitant appeler les fonctions

```
1 import folderName.subFolder.ClassName; // ClassName étant  
   le nom du fichier à importer et à répéter à chaque appel  
2 import folderName.subFolder.ClassName.*; // Attention :  
   importe tout "dans le fichier", risque d'écrasement !
```