La classe String

En Java, les chaînes de caractères sont des instances de la classe String. Les chaînes de caractères écrites littéralement sont toujours délimitées par des guillemets :

```
"Hello World"
```

String et tableau de caractères

Contrairement à d'autres langages de programmation, une chaîne de caractères ne peut pas être traitée comme un tableau. Si on souhaite accéder à un caractère de la chaîne à partir de son index, il faut utiliser la méthode String.charAt. On peut ainsi parcourir les caractères d'une chaîne :

```
String s = "Hello World";

for (int i = 0; i < s.length(); ++i) {
  char c = s.charAt(i);
  System.out.println(c);
}
```

La méthode String.length permet de connaître le nombre de caractères dans la chaîne. Il n'est malheureusement pas possible d'utiliser un for amélioré pour parcourir les caractères d'une chaîne car la classe String n'implémente pas l'interface Iterable. Par contre, il est possible d'obtenir un tableau des caractères avec la méthode String.toCharArray. On peut alors parcourir ce tableau avec un for amélioré.

```
String s = "Hello World";

for (char c : s.toCharArray()) {
  System.out.println(c);
}
```

> La méthode String.toCharArray a l'inconvénient de créer un tableau de la même longueur que la chaîne et de copier un à un les caractères. Si votre programme manipule intensivement des chaînes de caractères de taille importante, cela peut être pénalisant pour les performances. Depuis Java 8, il existe avec une nouvelle solution à ce problème avec un

impact mémoire quasi nul : l'utilisation des streams et des lambdas.

```
String s = "Hello World";
s.chars().forEach(c -> System.out.println((char)c));
```

Quelques méthodes utilitaires

Voici ci-dessous, quelques méthodes utiles fournies par la classe String. Reportez-vous à la documentation de la classe pour consulter la liste complète des méthodes.

String.equals

Compare la chaîne de caractères avec une autre chaînes de caractères.

```
System.out.println("a".equals("a"));   // true
System.out.println("a".equals("ab"));  // false
System.out.println("ab".equals("AB")); // false
```

String.equalsIgnoreCase

Comme la méthode précédente sauf que deux chaînes qui ne diffèrent que par la casse seront considérées comme identiques.

```
System.out.println("a".equalsIgnoreCase("a"));   // true
System.out.println("a".equalsIgnoreCase("ab"));  // false
System.out.println("ab".equalsIgnoreCase("AB")); // true
```

String.compareTo

Compare la chaîne de caractères avec une autre chaînes de caractères. La comparaison se fait suivant la taille des chaînes et l'ordre lexicographique des caractères. Cette méthode retourne 0 si les deux chaînes sont identiques, une valeur négative si la première est inférieure à la seconde et une valeur positive si la première est plus grande que la seconde.

```
System.out.println("a".compareTo("a"));   // 0
System.out.println("a".compareTo("ab"));  // < 0
System.out.println("ab".compareTo("a"));  // > 0
System.out.println("ab".compareTo("az")); // < 0
System.out.println("ab".compareTo("AB")); // > 0
```

String.compareToIgnoreCase

Comme la méthode précédente sauf que deux chaînes qui ne diffèrent que par la casse seront considérées comme identiques.

```
System.out.println("a".compareToIgnoreCase("a"));   // 0
System.out.println("a".compareToIgnoreCase("ab"));  // < 0
System.out.println("ab".compareToIgnoreCase("a"));  // > 0
System.out.println("ab".compareToIgnoreCase("az")); // < 0
System.out.println("ab".compareToIgnoreCase("AB")); // 0
```

String.concat

Concatène les deux chaînes dans une troisième. Cette méthode est équivalente à l'utilisation de l'opérateur **+**.

```
String s = "Hello".concat(" ").concat("World"); // "Hello World"
```

String.contains

Retourne **true** si la chaîne contient une séquence de caractères donnée.

```
boolean b = "Hello World".contains("World"); // true
b = "Hello World".contains("Monde");         // false
```

String.endsWith

Retourne **true** si la chaîne se termine par une chaîne de caractères donnée.

```
boolean b = "Hello World".endsWith("World"); // true
b = "Hello World".endsWith("Hello");         // false
```

String.startsWith

Retourne **true** si la chaîne commence par une chaîne de caractères donnée.

```
boolean b = "Hello World".endsWith("Hello"); // true
b = "Hello World".endsWith("World");         // false
```

String.isEmpty

Retourne **true** si la chaîne est la chaîne vide (*length()* vaut 0)

```
boolean b = "".isEmpty();    // true
b = "Hello World".isEmpty(); // false
```

String.length

Retourne le nombre de caractères dans la chaîne.

```
int n = "Hello World".length(); // 11
```

String.replace

Remplace un caractère par un autre dans une nouvelle chaîne de caractères.

```
String s = "Hello World".replace('l', 'x'); // "Hexxo Worxd"
```

Cette méthode est surchargée pour accepter des chaînes de caractères comme paramètres.

```
String s = "Hello World".replace(" World", ""); // "Hello"
```

String.substring

Crée une nouvelle sous-chaîne à partir de l'index de début et jusqu'à l'index de fin (non inclus).

```
String s = "Hello World".substring(2, 4); // "ll"
s = "Hello World".substring(0, 5);        // "Hello"
```

String.toLowerCase

Crée une chaîne de caractères équivalente en minuscules.

```
String s = "Hello World".toLowerCase(); // "hello world"
```

String.toUpperCase

Crée une chaîne de caractères équivalente en majuscules.

```
String s = "Hello World".toUpperCase(); // "HELLO WORLD"
```

String.trim

Crée une nouvelle chaîne de caractères en supprimant les espaces au début et à la fin.

```
String s = "       Hello World      ".trim(); // "Hello World"
```

Construction d’une instance de String

La classe String possède plusieurs constructeurs qui permettent de créer une chaîne de caractères avec l’opérateur **new**.

```
String s1 = new String(); // chaîne vide

String hello = "Hello World";
String s2 = new String(hello); // copie d'un chaîne

char[] tableau = {'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd'};
String s3 = new String(tableau); // à partir d'un tableau de caractères.

byte[] tableauCode = {72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100};
String s4 = new String(tableauCode); // à partir d'un tableau de code UTF-16
```

Immutabilité des chaînes de caractères

Les instances de la classe String sont immutables. Cela signifie qu’il est impossible d’altérer le contenu de la chaîne de caractères une fois qu’elle a été créée. Si vous reprenez la liste des méthodes ci-dessus, vous verrez que toutes les méthodes qui *modifient* le contenu de la chaîne de caractères crée une nouvelle chaîne de caractères et laissent intacte la chaîne d’origine. Cela signifie que des opérations intensives sur les chaînes de caractères peuvent être pénalisantes pour le temps d’exécution et l’occupation mémoire puisque toutes les opérations se font finalement par copie.

Nous avons vu qu’il n’existe pas réellement de constante en Java mais uniquement des attributs déclarés avec **static** et **final**. Cette immutabilité permet de garantir qu’une variable de String déclarée **static** et **final** ne peut plus être modifié.

La JVM tire également partie de cette immutabilité afin de réaliser des optimisations de place mémoire. Si par exemple vous écrivez plusieurs fois dans le code source la même chaîne de caractères, la JVM ne créera pas un nouvel emplacement mémoire pour cette chaîne. Ainsi, il est possible d’avoir des comportements assez déroutants au premier abord en Java :

```
String s = "test";

System.out.println(s == "test");                  // true
System.out.println(s == new String("test"));      // false
System.out.println(new String("test") == "test"); // false
```

Dans le code ci-dessus, on utilise l'opérateur **==** donc on ne compare pas le contenu des chaînes de caractères mais la référence des objets. La chaîne de caractères « test » apparaît plusieurs fois dans le code. Donc quand la JVM va charger la classe qui contient ce code, elle ne créera qu'une et une seule fois l'instance de String pour « test ». Voilà pourquoi la ligne 3 affiche **true**. Le contenu de la variable *s* référence exactement la même instance de String. Par contre, les lignes 4 et 5 créent explicitement une nouvelle instance de String avec l'opérateur **new**. Il s'agit donc de nouveaux objets avec de nouvelles références.

La classe StringBuilder

La classe StringBuilder permet de construire une chaîne de caractères par ajout (concaténation) ou insertion d'élements. Il est même possible de supprimer des portions. La quasi totalité des méthodes de la classe StringBuilder retourne l'instance courante du StringBuilder ce qui permet de chaîner les appels en une seule instruction. Pour obtenir la chaîne de caractères, il suffit d'appeler la méthode StringBuilder.toString.

```
StringBuilder sb = new StringBuilder();
sb.append("Hello")
  .append(" ")
  .append("world")
  .insert(5, " the")    // On insère la chaîne à l'index 5
  .append('!');
System.out.println(sb); // "Hello the world!"

sb.reverse();
System.out.println(sb); // "!dlrow eht olleH"

sb.deleteCharAt(0).reverse();
System.out.println(sb); // "Hello the world"
```

La classe StringBuilder permet de pallier au fait que les instances de la classe String sont immutables. D'ailleurs, l'opérateur **+** de concaténation de chaînes n'est qu'un sucre syntaxique, le

compilateur le remplace par une utilisation de la classe StringBuilder.

```
String s1 = "Hello";
String s2 = "the";
String s3 = "world";
String message = s1 + " " + s2 + " " + s3; // "Hello the world"
```

Le code ci-dessus sera en fait interprété par le compilateur comme ceci :

```
String s1 = "Hello";
String s2 = "the";
String s3 = "world";
String message = new StringBuilder().append(s1).append(" ").append(s2).append("
").append(s3).toString();
```

Formatage de texte

La méthode de classe String.format permet de passer une chaîne de caractères décrivant un formatage ainsi que plusieurs objets correspondant à des paramètres du formatage.

```
String who = "the world";
String message = String.format("Hello %s!", who);

System.out.println(message); // "Hello the world!"
```

Dans l'exemple ci-dessus, la chaîne de formatage « Hello %s » contient un paramètre identifié par *%s* (s signifie que le paramètre attendu est de type String.

Un paramètre dans la chaîne de formatage peut contenir différente information :

```
%[index$][flags][taille]conversion
```

L'index est la place du paramètre dans l'appel à la méthode String.format.

```
int quantite = 12;
LocalDate now = LocalDate.now();

String message = String.format("quantité = %1$010d au %2$te %2$tB %2$tY", quantite, now);
```

```
System.out.println(message); // "quantité = 0000000012 au 5 septembre 2017"
```

Il existe également une définition de la méthode String.format qui attend une instance de Locale en premier paramètre. La locale indique la langue du message et permet de formater les nombres, les dates, etc comme attendu.

```
int quantite = 12;
LocalDate now = LocalDate.now();

String message = String.format(Locale.ENGLISH, "quantity = %1$010d on %2$te %2$tB %2$tY",
quantite, now);

System.out.println(message); // "quantity = 0000000012 on 5 september 2017"
```

Pour mieux comprendre la syntaxe des paramètres dans une chaîne de formatage, reportez-vous à la documentation du Formatter qui est utilisé par la méthode String.format.

> Il est également possible de formater des messages avec la classe MessageFormat. Il s'agit d'une classe plus ancienne qui offre une syntaxe différente pour décrire les paramètres dans la chaîne de formatage.

Les expressions régulières

Certaines méthodes de la classe String acceptent comme paramètre une expression régulière (*regular expression* ou *regexp*). Une expression régulière permet d'exprimer avec des motifs un ensemble de chaînes de caractères possibles. Par exemple la méthode String.matches prend un paramètre de type String qui est interprété comme une expression régulière. Cette méthode retourne **true** si la chaîne de caractères est conforme à l'expression régulière passée en paramètre.

```
boolean match = "hello".matches("hello");
System.out.println(match); // true
```

L'intérêt des expressions régulières est qu'elles peuvent contenir des classes de caractères, c'est-à-dire des caractères qui sont interprétés comme représentant un ensemble de caractères.

Les classes de caractères dans une expression régulière

.	N'importe quel caractère

[abc]	Soit le caractère a, soit le caractère b, soit le caractère c
[a-z]	N'importe quel caractère de a à z
[^a-z]	N'importe quel caractère qui n'est pas entre a et z
\s	Un caractère d'espacement (espace, tabulation, retour à la ligne, retour chariot, saut de ligne)
\S	Un caractère qui n'est pas un caractère d'espacement (équivalent à [^\s]
\d	Un caractère représentant un chiffre (équivalent à [0-9]
\D	Un caractère ne représentant pas un chiffre (équivalent à [^0-9])
\w	Un caractère composant un mot (équivalent à [a-zA-Z_0-9]
\W	Un caractère ne composant pas un mot (équivalent à [^\w])

```
String s = "hello";
System.out.println(s.matches("....."));           // true
System.out.println(s.matches("h[a-m]llo"));       // true
System.out.println(s.matches("\\w\\w\\w\\w\\w")); // true
System.out.println(s.matches("h\\D\\S.o"));       // true
```

Une expression régulière peut contenir des quantificateurs qui permettent d'indiquer une séquence de caractères dans la chaîne.

Les quantificateurs dans une expression régulière

X?	X est présent zéro ou une fois
X*	X est présent zéro ou n fois
X+	X est présent au moins une fois
X{n}	X est présent exactement n fois
X{n,}	X est présent au moins n fois
X{n,m}	X est présent entre n et m fois

```
String s = "hello";
System.out.println(s.matches(".*"));                // true
System.out.println(s.matches(".+"));                // true
System.out.println(s.matches("X?hel+oW?"));         // true
System.out.println(s.matches(".+l{2}o"));           // true
System.out.println(s.matches("[eh]{0,2}l{1,100}o")); // true
```

> Il existe beaucoup d'autres motifs qui peuvent être utilisés dans une expression régulière. Reportez-vous à la documentation Java.

Il est possible d'utiliser la méthode String.replaceFirst ou String.replaceAll pour remplacer respectivement la première ou toutes les occurrences d'une séquence de caractères définie par une expression régulière.

```
String s = "hello";
System.out.println(s.replaceAll("[aeiouy]", "^_^")); // h^_^ll^_^
```

La méthode String.split permet de découper une chaîne de caractères en tableau de chaînes de caractère en utilisant une expression régulière pour identifier le séparateur.

```
String s = "hello the world";

// ["hello", "the", "world"]
String[] tab = s.split("\\W");

// ["hello", "world"]
tab = s.split(" the ");

// ["he", "", "", "the w", "r", "d"]
tab = s.split("[ol]");
```

> Les expressions régulières sont représentées en Java par la classe Pattern. Il est possible de créer des instances de cette classe en compilant une expression régulière à l'aide de la méthode de classe Pattern.compile.

Revision #1
Created 10 February 2025 12:15:00 by Nicolas
Updated 10 February 2025 12:22:19 by Nicolas