Le type `String`

Les chaînes de caractères Java sont définies par le type **String**.
En toute rigueur, ce n'est pas un type comme les types élémentaires mais une classe.

Syntaxe : déclaration d'une chaîne de caractères

`String` *identificateur*;

L'initialisation peut se faire en affectant à la variable un littéral de type `String`

Exemples : Déclaration et initialisation

```
String unNom;
String message = "Bonjour tout le monde !";
```

Le type `char`

Les caractères (constituants d'une chaîne) peuvent aussi se représenter en tant que tels :

☞ le type `char`

- Le caractère s'écrit entre guillemets simples
- un `char` contient exactement 1 caractère

```
char c1 = 'm';
char c2 = 'M';
char c3 = ' '; // espace
char c5 = '2';
```

Le type `char` (2)

Un **caractère précédé par un « backslash »** (\) a une signification spéciale :

- Caractère spécial :

```
char c5 = '\n';  // Saut de ligne
char c6 = '\t';  // tabulateur
```

- Caractère qui risque d'être mal interprété :

```
char c7 = '\'';  //guillemet simple
char c8 = '\\'; //backslash
```

- Sinon, le « backslash » est erroné :

```
char c9 = '\a';
```

`Error: Invalid escape character`

`String` : Sémantique des opérateurs = et ==

Comme pour les tableaux, une variable de type `String` contient une **référence** vers une chaîne de caractères. La sémantique des opérateurs = et == est donc la même que pour les tableaux :

```
String chaine = "";     // chaine pointe vers ""
String chaine2 = "foo"; // chaine2 pointe vers "foo"
chaine = chaine2 ;      // chaine et chaine2 pointent vers "foo"
(chaine == chaine2)     // retourne true
```

`String` : Sémantique des opérateurs = et ==

Les litteraux de type `String` occupent une zone mémoire unique

☞ « Pool » des littéraux

```
String chaine1 = "foo";  // chaine1 pointe vers le litteral "foo"
String chaine2 = "foo" ; // chaine2 pointe vers le litteral "foo"
if (chaine1 == chaine2 ) // true : chaine1 et chaine2 contiennent la même
                         //adresse
```

`String` : Affichage

Qu'affiche le code suivant ?

```
String chaine = "Welcome";
System.out.print(chaine);
```

Puisque la variable `chaine` contient une référence à la zone mémoire contenant la chaîne `"Welcome"`, il est raisonnable de penser que ce code affiche une adresse (comme pour les tableaux de manière générale) : **ce n'est pas le cas !**

☞ Le code précédent affiche `Welcome`

☞ Pour les `String` l'affichage est défini de sorte à prendre en compte la référence pointée plutôt que la référence elle-même. C'est une exception.

Concaténation

`chaine1 + chaine2` produit une **nouvelle chaîne** associée à la valeur littérale constituée de la **concaténation** des valeurs littérales de `chaine1` et de `chaine2`.

Exemple : constitution du nom complet à partir du nom de famille et du prénom :

```
String nom;
String prenom;
....
nom = nom + " " + prenom;
```

Important ! La concaténation **ne modifie jamais** les chaînes concaténées. Elle effectue une **copie** de ces chaînes dans une autre zone en mémoire.

Concaténation

Les combinaisons suivantes sont possibles pour la concaténation de deux chaînes :

`String + String`

`String + typeDeBase` `typeDeBase + String`

où `String` correspond à une variable ou une valeur littérale de type `String`, et `typeDeBase` à une variable ou une valeur littérale de l'un des types de base (`char`, `boolean`, `double`, `int` etc.).

Exemple revisité (avec `char`) :

```
String nom;
String prenom;
....
nom = nom + ' ' + prenom;
```

Concaténation

Les concaténations de la forme `String+char` constituent donc un moyen très pratique pour ajouter des caractères à la fin d'une chaîne.

De même la concaténation `char+String` permet l'ajout d'un caractère en début de chaîne.
Exemple : ajout d'un 's' final au pluriel :

```
String reponse = "solution";
//...
if (n > 1) {
   reponse = reponse + 's';
}
```

(Non-)Egalité de `Strings`

Les opérateurs suivants :

`==`	égalité
`!=`	non-égalité

testent si deux variables `String` **font référence** (ou non) à la même zone mémoire (occupée par une chaîne de caractères).

Ceci est le cas lorsque les variables de types `String` ont été initialisées au moyen de **littéraux**

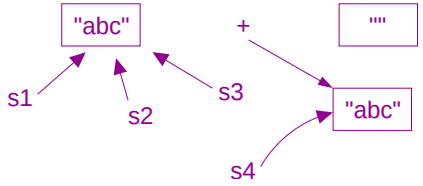
Exemple : utilisation de l'opérateur `!=`

```
while (reponse != "oui") ....;
```

(Non-)Egalité de `Strings` (2)

```
String s1 = "abc";    // s1 pointe vers le littéral "abc"
String s2 = "abc";    // idem (donc même zone mémoire que s1)
String s3 = s2;       // s3 stocke la même adresse que s2
String s4 = s1 + "";  // s4 contient l'adresse d'une nouvelle chaîne
                      // (construite par concaténation)
System.out.println((s1==s2) && (s2==s3)); // affiche true
System.out.println(s4);                   // affiche abc
System.out.println((s1==s4));             // affiche false
```

Situation en mémoire :



Comment faire pour **comparer les contenus référencés** plutôt que les références ?

☞ **Traitement spécifique** aux `String`

Comparaison de `String`

`chaine1.equals(chaine2)` teste si les chaînes de caractères référencées par `chaine1` et `chaine2` sont constituées des mêmes caractères

```
String s1 = "abc";
String s2 = "aBc";
String s4 = s1 + "";

System.out.println(s1.equals(s4)); // true
System.out.println(s1.equals(s2)); // false
```