Chapitre 2

Structures de programmation de base (2)

Thibaud Martinez

thibaud.martinez@dauphine.psl.eu

Dauphine | PSL
UNIVERSITÉ PARIS

Chaînes de caractères

- Les chaînes de caractères Java sont des séquences de caractères Unicode.
- Le type chaîne de caractères ne fait pas partie des types élémentaires.

```
String s = ""; // une chaîne de caractères vide
String message = "Bonjour";
```

> Pour en savoir plus sur Unicode, la lecture de cet article (en anglais) est vivement recommandée.
> The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)

Sous-chaîne de caractères

```
String msg = "Bonjour";
String s = msg.substring(0, 3); // s vaut "Bon"
```

Les caractères de la chaîne sont numérotés à partir de `0`. Ainsi, le dernier élément est à l'indice `longueur de la chaîne - 1`.

Concaténation

Il s'agit de mettre bout à bout deux chaînes de caractères ou plus. Pour cela on utilise l'opérateur `+`.

```
String prenom = "John";
String nom = "Snow";
int age = 25;
String message = prenom + " " + nom + ", " + age + " ans"
```

Immutabilité des chaînes de caractères

Une fois une chaîne de caractères créée, il n'est plus possible de la modifier.

Ainsi, lorsqu'on soufaite modifier une chaîne, il faudra en créer une nouvelle pour **remplacer** l'existante.

```
String msg = "Bonjour";
msg = msg.substring(0, 3) + "ne journée";
```

ℹ️ Puisque les chaînes de caractères sont immutables, les méthodes qui opérent sur des chaîne renvoient une valeur mais ne modifient pas la chaîne.

Tester l'égalité des chaînes de caractères

```
String s1 = "Hello";
String s2 = "Bonjour";

s1.equals(s2);           // false
"Bonjour".equals(s2);    // true
```

⚠️ **N'utilisez pas l'opérateur `==` pour tester si deux chaînes de caractères sont égales** ! Il détermine uniquement si les chaînes sont stockées au même endroit ou non. Bien sûr, si les chaînes de caractères se trouvent au même endroit, elles doivent être égales. Mais il est tout à fait possible de stocker des copies de chaînes identiques à des endroits différents.

Chaîne de caractère vide

La chaîne vide `""` est une chaîne de longueur 0.

On peut tester si une chaîne `str` est vide de la façon suivante :

```
if (str.equals(""))
```

ou

```
if (str.length() == 0)
```

La méthode `length` renvoie la taille d'une chaîne de caractères (plus exactement, le nombre de *code units* dans la chaîne).

Tableaux (*Arrays*)

Les tableaux sont séquences de valeurs de **même type**. Ils ont une **taille fixe**.

Déclaration

```
<type des éléments>[] <nom>;
```

```
int[] t;    // déclare un tableau d'entiers
```

Initialisation

```
new <type des éléments>[<taille du tableau>]
```

```
t = new int[100];           // initialisation d'un tableau de taille 100
int[] t2 = new int[100];    // déclaration-initialisation
```

Valeurs initiales par défaut

- Lorsqu'on initialise un tableau de **nombres**, tous les éléments sont initialisés avec `0` ;
- les tableaux de **booléens** sont initialisés avec `false` ;
- les tableaux d'**objets** sont initialisés avec `null` .

Créer un tableau avec des valeurs initiales

```
int[] fibonacci = {0, 1, 1, 2, 3, 5, 8, 13};
String[] realisateurs = {
    "Steven Spielberg",
    "Georges Lucas",
    "Francis Ford Coppola",
}
```

Accéder aux éléments d'un tableau

Les éléments du tableau sont numérotés à partir de `0`. Ainsi, le dernier élément est à l'indice `taille - 1`.

L'accès à un élément se fait par son indice.

```
int[] fibonacci = {0, 1, 1, 2, 3, 5, 8, 13};
int element = fibonacci[3];      // element vaut 2
```

De la même manière, avec `[ ]`, on peut modifier un élément situé à un index donné.

```
int[] t = new int[100];
for (int i = 0; i < 100; i++) {
    t[i] = i; // rempli le tableau avec les nombres de 0 à 99
}
```

Taille (*length*)

Pour trouver la taille d'un tableau, on utilise `.length`.

```
int[] tableau = new int[50];
tableau.length    // vaut 50
```

Boucle *"for each"*

On peut parcourir les éléments d'un tableau sans se soucier des indices grâce à une boucle *"for each"*.

```
String[] noms = {"Natalie", "Steve", "James", "Elizabeth"};

for (var n: noms) {
    System.out.println(n);  // affiche "Natalie Steve James Elizabeth"
}
```

Copier un tableau

⚠️ Lorsqu'on affecte la valeur d'une variable tableau à une autre variable, les deux variables feront référence au même tableau.

```
int[] nombres = {1, 2, 3, 4, 5, 6, 7, 8};
int[] selection = nombres;
selection[4] = 10;  // nombres[4] vaut désormais également 10
```

Pour copier toutes les valeurs du tableau, et non uniquement la référence vers le tableau, on utilisera la méthode `copyOf`.

```
selection = Arrays.copyOf(nombres, nombres.length)
```

Types énumérés

Parfois, une variable ne doit contenir qu'un ensemble restreint de valeurs. Supposons par exemple, qu'on vend des pizzas en quatre tailles : *small*, *medium*, *large* et *extra large*.

Un type énuméré a un **nombre fini de valeurs nommées**.

```
enum Taille { SMALL, MEDIUM, LARGE, EXTRA_LARGE };
```

On peut ensuite déclarer des variables de ce type.

```
Taille t = Taille.MEDIUM;
```

switch

Les expressions *switch* se prêtent particulièrement bien aux types énumérés.

```
var taille = Taille.SMALL;

String prix = switch(taille) {
    case Taille.SMALL           -> "10 €";
    case Taille.MEDIUM          -> "15 €";
    case Taille.LARGE           -> "20 €";
    case Taille.EXTRA_LARGE     -> "25 €";
};
```

Entrées/sorties

On souhaite pouvoir interagir avec des systèmes extérieurs au programme, tel que l'utilisateur, au travers de la console, ou encore le système de fichiers.

Lire depuis la console

Lire **une ligne de texte** depuis la console (entrée standard) se fait de la façon suivante :

```
Scanner in = new java.util.Scanner(System.in);
System.out.print("Quel est votre nom ? ");
String nom = in.nextLine();
```

Pour lire **un unique mot délimité par des espaces**.

```
String prenom = in.next();
```

Pour lire un entier.

```
System.out.print("Quel age avez-vous ? ");
int age = in.nextInt();
```

Écrire sur la console

Pour afficher du texte sur la console (sortie standard).

```
System.out.print("Texte sans retour à la ligne");
System.out.println("Texte avec retour à la ligne");
```

Pour afficher un message textuel contenant des variables.

```
double doubleVar = 10.014546546;
int intVar = 9;
String stringVar = "exemple";

System.out.printf(
    "valeur double : %.2f, valeur int : %d, valeur string : %s\n",
    doubleVar, intVar, stringVar
);
```

ℹ️ Pour en savoir plus : Formatting Numeric Print Output.

Lire et écrire des fichiers

Lire

Pour lire un fichier, on utilise là encore un `Scanner`.

```
java.util.Scanner in = new java.util.Scanner(
    java.nio.file.Path.of("myfile.txt"),
    java.nio.charset.StandardCharsets.UTF_8
);
String premiereLigne = in.nextLine();
```

Écrire

Pour écrire dans un fichier, on utilise un `PrintWriter`.

```
java.io.PrintWriter out = new java.io.PrintWriter(
    "myfile.txt",
    java.nio.charset.StandardCharsets.UTF_8
);
out.println("Ceci est une ligne écrite dans le fichier");
```

Si le fichier n'existe pas, il est créé. On peut utiliser les commandes `print`, `println` et `printf` comme pour l'affichage sur `System.out`.