Les tableaux

Les tableaux représentent des collections de valeurs ou d'objets. En Java, les tableaux sont eux-mêmes des objets. Donc une variable de type tableau peut avoir la valeur **null**. Une variable de type tableau se déclare en ajoutant des crochets à la suite du type :

```
int[] tableau;
```

Il est également possible de placer les crochets après le nom de la variable :

```
int tableau[];
```

Initialisation

Il est possible d'initialiser une variable de type tableau à partir d'une liste fixe délimitée par des accolades.

```
int[] tableauEntier = {1, 2, 3, 4, 5};
String[] tableauChaine = {"Bonjour", "le", "monde"};
```

Création avec new

Les tableaux étant des objets, il est également possible de les créer avec le mot-clé **new**.

```
int[] tableauEntier = new int[] {1, 2, 3, 4};
String[] tableauChaine = new String[] {"Bonjour", "le", "monde"};
```

Si on ne souhaite pas donner de valeurs d'initialisation pour les élements du tableau, il suffit d'indiquer uniquement le nombre d'éléments du tableau entre crochets.

```
int[] tableauEntier = new int[5];
String[] tableauChaine = new String[3];
```

Dans ce cas, les éléments d'un tableau sont tout de même initialisés avec une valeur par défaut (comme pour un attribut) :

Valeur par défaut d'un élément d'un tableau

Type	Valeur d'initialisation
boolean	false
char	`'\0'`
byte	0
short	0
int	0
long	0
float	0.0
double	0.0
référence d'objet	null

La taille du tableau peut être donnée par une constante, une expression ou une variable.

```
int t = 6;
int[] tableau = new int[t * t * 2];
```

Par contre, la taille d'un tableau est donné à sa création et ne peut plus être modifiée. Il n'est donc pas possible d'ajouter ou d'enlever des éléments à un tableau. Dans ce cas, il faut créer un nouveau tableau avec la taille voulue et copier le contenu du tableau d'origine vers le nouveau tableau.

Un tableau dispose de l'attribut **length** permettant de connaître sa taille. L'attribut **length** ne peut pas être modifié.

```
int t = 6;
int[] tableau = new int[t * t * 2];
System.out.println(tableau.length); // 72
```

> Il est tout à fait possible de créer un tableau vide, c'est-à-dire avec une taille de zéro.

```
int[] tableau = new int[0];
```

Par contre, donner une taille négative est autorisé par le compilateur mais aboutira à une erreur d'exécution avec une exception de type java.lang.NegativeArraySizeException.

Accès aux éléments d'un tableau

L'accès aux éléments d'un tableau se fait en donnant l'indice d'un élément entre crochets. Le premier élément d'un tableau a l'indice **0**. Le dernier élément d'un tableau a donc comme indice la taille du tableau moins un.

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

int premierElement = tableau[0];
int dernierElement = tableau[tableau.length - 1];

System.out.println(premierElement); // 1
System.out.println(dernierElement); // 5

for (int i = 0, j = tableau.length - 1; i < j; ++i, --j) {
  int tmp = tableau[j];
  tableau[j] = tableau[i];
  tableau[i] = tmp;
}
```

Comme le montre l'exemple précédent, il est bien sûr possible de parcourir un tableau à partir d'un indice que l'on fait varier à l'aide d'une boucle **for**. Mais il est également possible de parcourir tous les élements d'un tableau avec un **for** amélioré.

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

for (int v : tableau) {
  System.out.println(v);
}
```

L'utilisation d'un **for** amélioré est préférable lorsque cela est possible. Par contre, il n'est pas possible avec un **for** amélioré de connaître l'indice de l'élément courant.

Si le programme tente d'accéder à un indice de tableau trop grand (ou un indice négatif), une erreur de type java.lang.ArrayIndexOutOfBoundsException survient.

```
int[] tableau = {1, 2, 3, 4, 5};
int value = tableau[1000]; // ERREUR À L'EXÉCUTION
```

Tableau multi-dimensionnel

Il est possible d'initialiser un tableau à plusieurs dimensions.

```
int[][] tableauDeuxDimensions = {{1, 2}, {3, 4}};

int[][][] tableauTroisDimensions = {{{1, 2}, {3, 4}}, {{5, 6}, {7, 8}}};

System.out.println(tableauDeuxDimensions[0][1]);
System.out.println(tableauTroisDimensions[0][1][0]);
```

Il est également possible de créer un tableau multi-dimensionnel avec le mot-clé **new**.

```
int[][] tableauDeuxDimensions = new int[2][10];
int[][][] tableauTroisDimensions = new int[2][10][5];
```

Il n'existe pas réellement de type tableau multi-dimensionnel. Le compilateur le traite comme un tableau de tableaux. Il est donc autorisé de déclarer des tableaux sans préciser les dimensions au delà de la première et d'affecter ensuite des tableaux à chaque valeur. Ces tableaux peuvent d'ailleurs avoir des tailles différentes.

```
int[][] tableauDeuxDimensions = new int[2][];

tableauDeuxDimensions[0] = new int[10];
tableauDeuxDimensions[1] = new int[5];
```

Conversion en chaîne de caractères

Si vous affichez un tableau sur la sortie standard, vous serez certainement surpris.

```
int[] tableau = {1, 2, 3, 4, 5};
System.out.println(tableau);
```

La code précédent affichera sur la sortie standard quelque chose comme ceci :

```
[I@ee7d9f1
```

Cela peut sembler un bug mais il n'en est rien. En fait, la conversion d'un objet en chaîne de caractères affiche par défaut son type suivi du caractère @ suivi du code de hachage de l'objet. Normalement le type d'un objet correspond au nom de sa classe. Mais le type d'un tableau est noté **[** suivi du type des éléments du tableau (**I** indique le type primitif **int**).

Pour obtenir une chaîne de caractères donnant le contenu du tableau, il faut utiliser la classe outil java.util.Arrays qui contient des méthodes de classe **toString** adaptées pour les tableaux.

```
int[] tableau = {1, 2, 3, 4, 5};
System.out.println(java.util.Arrays.toString(tableau));
```

> Pour les tableaux multi-dimensionnels, vous pouvez utiliser la méthode java.util.Arrays.deepToString(Object[]).

Égalité de deux tableaux

En Java, il n'est pas possible d'utiliser l'opérateur **==** pour comparer deux objets. En effet, cet opérateur compare la référence des variables. Cela signifie qu'il indique **true** uniquement si les deux variables référencent le même objet.

```
int[] tableau1 = {1, 2, 3, 4, 5};
int[] tableau2 = {1, 2, 3, 4, 5};

System.out.println(tableau1 == tableau1); // true
System.out.println(tableau1 == tableau2); // false
```

Pour comparer deux objets, il faut utiliser la méthode **equals**. Les tableaux en Java disposent de la méthode **equals**, malheureusement, elle a exactement le même comportement que l'utilisation de l'opérateur **==**.

```
int[] tableau1 = {1, 2, 3, 4, 5};
int[] tableau2 = {1, 2, 3, 4, 5};

System.out.println(tableau1.equals(tableau1)); // true
System.out.println(tableau1.equals(tableau2)); // false
```

La classe outil java.util.Arrays fournit des méthodes de classe **equals** pour comparer des tableaux en comparant un à un leurs éléments.

```
int[] tableau1 = {1, 2, 3, 4, 5};
int[] tableau2 = {1, 2, 3, 4, 5};

System.out.println(java.util.Arrays.equals(tableau1, tableau1)); // true
System.out.println(java.util.Arrays.equals(tableau1, tableau2)); // true
```

Il est également possible de comparer des tableaux d'objets. Dans ce cas, la comparaison des élements se fait en appelant la méthode **equals** de chaque objet. La méthode **equals** possède la

signature suivante :

```
public boolean equals(Object obj) {
  // ...
}
```

Par exemple, la classe java.lang.String fournit une implémentation de la méthode **equals**. Il est donc possible de comparer des tableaux de chaînes de caractères.

```
String[] tableau1 = {"premier", "deuxième", "troisième", "quatrième"};
String[] tableau2 = {"premier", "deuxième", "troisième", "quatrième"};

System.out.println(java.util.Arrays.equals(tableau1, tableau2)); // true
```

> Pour les tableaux multi-dimensionnels, vous pouvez utiliser la méthode java.util.Arrays.deepEquals(Object[], Object[])

Tri & recherche

Le tri et la recherche sont des opérations courantes sur des tableaux de valeurs. La classe outil java.util.Arrays offrent un ensemble de méthodes de classe pour nous aider dans ces opérations.

Tout d'abord, java.util.Arrays fournit plusieurs méthodes **sort**. Celles prenant un tableau de primitives en paramètre trient selon l'ordre naturel des éléments.

```
int[] tableau = {1, 5, 4, 3, 2};
java.util.Arrays.sort(tableau);
System.out.println(java.util.Arrays.toString(tableau));
```

Il est également possible de trier certains tableaux d'objets. Par exemple, il est possible de trier des tableaux de chaînes de caractères.

```
String[] tableau = {"premier", "deuxième", "troisième", "quatrième"};
java.util.Arrays.sort(tableau);
System.out.println(java.util.Arrays.toString(tableau));
```

> La méthode java.util.Arrays.sort(Object[]) permet de trier des tableaux d'objets dont la classe implémente l'interface java.lang.Comparable.

java.util.Arrays fournit des méthodes **binarySearch** qui implémentent l'algorithme de recherche binaire. Ces méthodes attendent comme paramètres un tableau et une valeur compatible avec le type des éléments du tableau. Ces méthodes retournent l'index de la valeur trouvée. Si la valeur n'est pas dans le tableau, alors ces méthodes retournent un nombre négatif. La valeur absolue de ce nombre correspond à l'index auquel la valeur aurait dû se trouver plus un.

```
int[] tableau = {10, 20, 30, 40, 50};
System.out.println(java.util.Arrays.binarySearch(tableau, 20)); // 1
System.out.println(java.util.Arrays.binarySearch(tableau, 45)); // -5
```

> L'algorithme de recherche binaire ne fonctionne correctement que pour un tableau trié.

Copie d'un tableau

Comme il n'est pas possible de modifier la taille d'un tableau, la copie peut s'avérer une opération utile. java.util.Arrays fournit des méthodes de classe *copyOf* et *copyOfRange* pour réaliser des copies de tableaux.

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

int[] nouveauTableau = java.util.Arrays.copyOf(tableau, tableau.length - 1);
System.out.println(java.util.Arrays.toString(nouveauTableau)); // [1, 2, 3, 4]

nouveauTableau = java.util.Arrays.copyOf(tableau, tableau.length + 1);
System.out.println(java.util.Arrays.toString(nouveauTableau)); // [1, 2, 3, 4, 5, 0]

nouveauTableau = java.util.Arrays.copyOfRange(tableau, 2, tableau.length);
System.out.println(java.util.Arrays.toString(nouveauTableau)); // [3, 4, 5]

nouveauTableau = java.util.Arrays.copyOfRange(tableau, 2, 3);
System.out.println(java.util.Arrays.toString(nouveauTableau)); // [3]
```

Pour réaliser une copie, il existe également la méthode java.lang.System.arraycopy. Contrairement aux précédentes, cette méthode ne crée pas de nouveau tableau, elle copie d'un tableau existant vers un autre tableau existant.

```
int[] tableau = {1, 2, 3, 4, 5};
int[] destination = new int[3];
```

```
/* Les paramètres attendus sont :
 * - le tableau source
 * - l'index de départ dans le tableau source
 * - le tableau destination
 * - l'index de départ dans le tableau destination
 * - le nombre d'éléments à copier
 */
System.arraycopy(tableau, 1, destination, 0, destination.length);
System.out.println(java.util.Arrays.toString(destination)); // [2, 3, 4]
```

Typage d’un tableau

Un tableau est un objet. Cela implique qu’il respecte les règles de typage du langage. Ainsi on ne peut mettre dans un tableau que des valeurs qui peuvent être affectées au type des éléments

```
String[] tableau = new String[10];
tableau[9] = "Bonjour"; // OK
tableau[8] = new Voiture(); // ERREUR DE COMPILATION
```

De plus, les tableaux peuvent être affectés à des variables dont le type correspond à un tableau d’éléments de type parent.

```
Integer[] tableau = {1, 2, 3, 4};
Number[] tableauNumber = tableau;
```

Pour l’exemple précédent, il faut se rappeler la classe enveloppe java.lang.Integer hérite de la classe java.lang.Number. Cependant, un tableau conserve son type d’origine : si on affecte une valeur dans un tableau, elle doit non seulement être compatible avec le type de la variable (pour passer la compilation) mais aussi être compatible avec le type de tableau à l’exécution. Si cette dernière condition n’est pas remplie, on obtiendra une erreur de type java.lang.ArrayStoreException au moment de l’exécution.

```
Integer[] tableau = {1};
Number[] tableauNumber = tableau;
tableauNumber[0] = Float.valueOf(2.3f); // ERREUR À L'EXÉCUTION
```

Conversion d’un tableau en liste

La plupart des API Java utilisent des collections plutôt que des tableaux. Pour transformer un tableau d'objets en liste, on utilise la méthode java.util.Arrays.asList. La liste obtenue possède une taille fixe. Par contre le contenu de la liste est modifiable, et toute modification des éléments de cette liste sera répercutée sur le tableau.

```
String[] tableau = {"Bonjour", "le", "monde"};
java.util.List<String> liste = java.util.Arrays.asList(tableau);

liste.set(0, "Hello");
liste.set(1, "the");
liste.set(2, "world");

// Le tableau a été modifié à travers la liste
System.out.println(java.util.Arrays.toString(tableau)); // [Hello, the, world]
```

Revision #1
Created 2025-02-10 11:59:45 UTC by Nicolas
Updated 2025-02-10 12:13:14 UTC by Nicolas