Tableaux

Sommaire

1 Définition

Un tableau est une structure de données contenant un groupe d'éléments tous du même type. Le type des éléments peut être un type primitif ou une classe. Lors de la définition d'un tableau les [] spécifiant qu'il s'agit d'un tableau peuvent être placés avant ou après le nom du tableau :

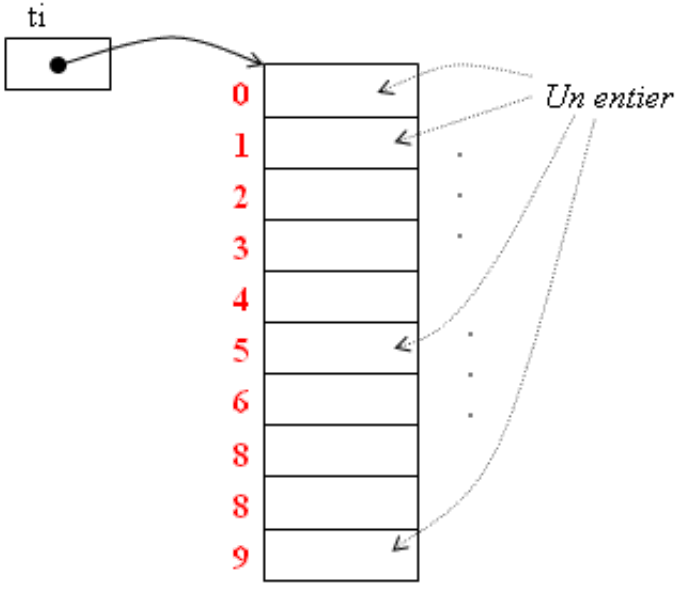
```
// ti est un tableau à une dimension de int
int ti [];
// tc est un tableau à une dimension de char
char[] tc;
```

Un tableau peut être initialisé :

```
int ti1 [] = { 1, 2, 3 , 4};
char[] tc = {'a', 'b', 'c'};
```

Pour allouer l'espace nécessaire au tableau il faut utiliser new :

```
// ti est un tableau à une dimension de 10 int
ti = new int [10];
// tc est un tableau à une dimension de 15 char
tc = new char [15];
```



L'utilisation d'un tableau pour lequel l'espace n'a pas été alloué provoque la levée d'une exception `NullPointerException`

Les éléments d'un tableau sont indicés à partir de 0. Chaque élément peut être accédé individuellement en donnant le nom du tableau suivi de l'indice entre [ ]

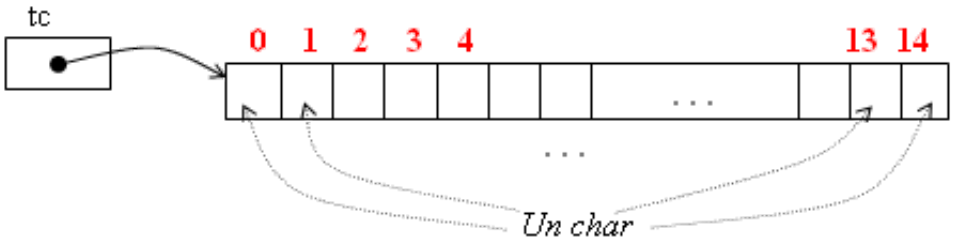
```
ti [0] ;// premier élément du tableau ti
ti [9] ;// dernier élément du tableau ti
```

L'accès à un élément du tableau en dehors des bornes provoque la levée d'une

exception `ArrayIndexOutOfBoundsException`

Un tableau possède un attribut length qui permet de connaître le nombre d'éléments d'un tableau.

- `ti.length` vaut 10.
- `tc.length` vaut 15.



2 Tableau à plusieurs dimensions

Lors d'une définition de tableau le nombre de crochets indique le nombre de dimensions du tableau.

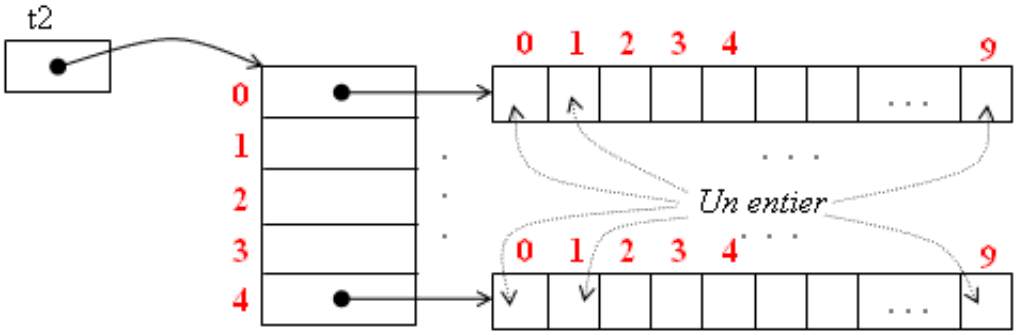
```
int t2[][] = new int[5][10] ;
```

définit un tableau à 2 dimensions 5 lignes sur 10 colonnes (ou l'inverse).

`t2[i]` désigne le $i^{ème}$ tableau à une dimension d'entiers.

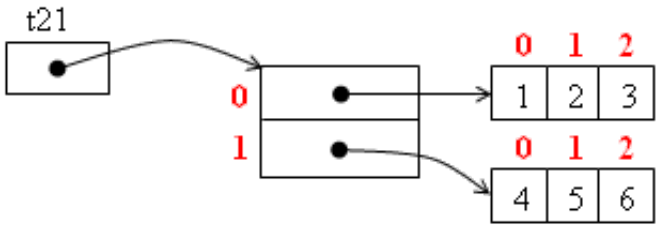
Un tableau à plusieurs dimensions peut être initialisé :

```
int t21[][] = {{1, 2, 3},
```



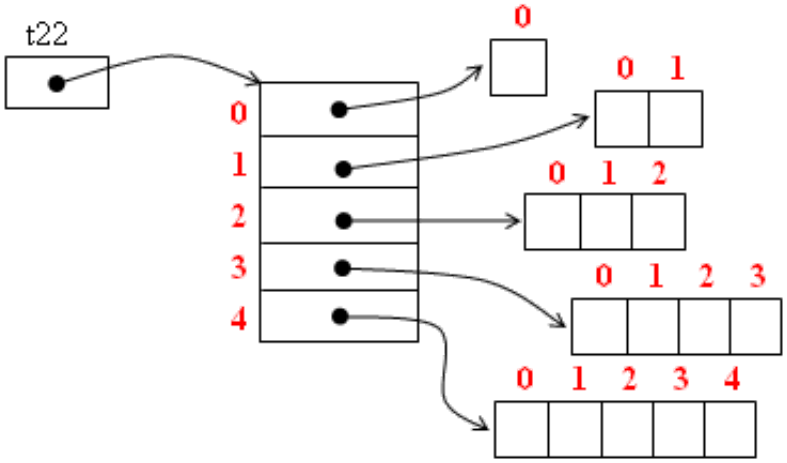
```
{4, 5, 6}};
```

définit un tableau dont la première dimension va de l'indice 0 à l'indice 1 et la deuxième dimension de l'indice 0 à l'indice 2.



Toutes les lignes d'un tableau à 2 dimensions n'ont pas forcément le même nombre d'éléments :

```
int t22[][] ;
t22 = new int[5][];
for( int i = 0; i< t22.length; ++i){
   t22[i]= new int [i+1];
}
for( int i = 0; i< t22.length; ++i){
   for( int j = 0; j<t22[i].length; ++j){
      //accès à t22[i][j]
   }
}
```



3 Tableaux en paramètre :

La spécification d'un paramètre tableau se fait en écrivant autant de couples de [] que de dimensions du tableau, mais sans donner la taille de chaque dimension. Cette taille peut être obtenue dans la méthode à l'aide de l'attribut length.

Le contenu des éléments peut être modifié, mais pas le tableau lui-même.

4 Utilitaires pour les tableaux

System.arraycopy : la classe System a une méthode de copie rapide de tableaux :

```
public static void arraycopy(Object src, int srcPos, Object dest, int destPos, int l)
```

Copie un tableau depuis src, et partir de la position srcPos, dans la destination dest à partir de destPos et sur une longueur l. La copie n'est pas une copie profonde : seules les références sont copiées.

- Si src et dest sont le même tableau, tout se passe comme si les éléments à copier étaient d'abord copiés dans un tableau auxiliaire.
- Si src ou dest vaut null, une NullPointerException est levée
- Une ArrayStoreException est levée si :
  - src ou dest n'est pas un tableau
  - src et dest sont des tableaux dont les éléments ne sont pas de même type.
- Une IndexOutOfBoundsException est levée si :
  - srcPos ou destPos ne sont pas « corrects »
  - destPos+l ou srcPos+l sortent du tableau

La classe java.lang.reflect.Array contient des méthodes statiques permettant :

- de récupérer un élément d'un tableau, en le convertissant dans un autre type
- d'affecter un tableau avec un élément
- de créer, à l'exécution, un tableau

`static Object get(Object t, int index)`	retourne l'objet d'indice index dans le tableau t. Cet élément est convertit en un des types enveloppants s'il est de type primitif (Integer pour int, Long pour long, etc...).
`static xxx getXxx(Object t, int index)`	retourne l'objet d'indice index dans le tableau t. Cet élément est convertit en le type primitif xxx si c'est possible
`static void set(Object t, int index, Object valeur)`	affecte une nouvelle valeur à l'élément d'indice index du tableau t. La valeur est convertie en le bon type avant l'affectation
`static void setXxx(Object t, int index, xxx z)`	affecte une nouvelle valeur à l'élément d'indice index du tableau t.

Les méthodes précédentes peuvent lever les exceptions suivantes :

- NullPointerException si l'objet au rang index vaut null.
- IllegalArgumentException si le premier argument n'est pas un tableau.
- ArrayIndexOutOfBoundsException si index est en dehors des bornes du tableau

`static Object newInstance(Class componentType, int l)`	Crée un nouveau tableau dont les éléments sont de type componentType et la longueur est l. Cette méthode peut lever les exceptions : • NullPointerException si le componentType vaut null. • nullNegativeArraySizeException si l est négative.
`static Object newInstance(Class componentType, int[] dimensions)`	Crée un nouveau tableau dont les éléments sont de type componentType : • Si componentType est une class ou une interface (pas tableau) le nouveau tableau a dimensions.length dimensions. Le nombre d'éléments dans la $i^{ème}$ dimension est égal à dimensions[i]. • Si componentType représente une classe tableau, le nombre de dimensions du nouveau tableau est égal à la somme du nombre de dimensions de componentType plus dimensions.length .

Exemple :

```
int dims[]={2, 3};
Integer ii = new Integer(1);
Object t = java.lang.reflect.Array.newInstance(ii.getClass(), dims);
```

Permet de créer un tableau d'Integer à 2 dimensions : 2 lignes et 3 colonnes.