chapitre 6: Tableaux et Vecteurs

Présentation pour

Java Software Solutions

Foundations of Program Design

Second Edition

by John Lewis et William Loftus

Java Software Solutions is published by Addison-Wesley

Tableaux et Vecteurs

- **tableaux et vecteurs sont des objets qui nous aident à organiser de grosse quantité d'informations**

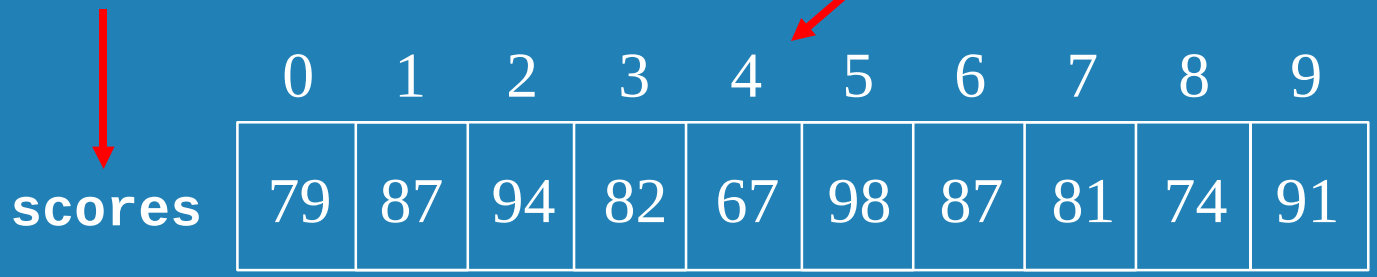
- **Le chapitre 6 se concentre sur:**
  - **déclaration de tableau et utilité**
  - **tableaux d'objets**
  - **triés les éléments d'un tableau**
  - **tableaux multidimensionnels**
  - **la classe `vector`**
  - **utilisation de tableaux pour gérer des graphiques**

Tableaux

- **Un *tableau* est une liste ordonnée de valeurs**

le tableau entier a un nom unique

chaque valeur possède un *indice*



Un tableau de taille N est indexé de zéro à N-1

Ce tableau contiens 10 valeurs indexées de 0 to 9

Tableaux

- **Une valeur particulière dans un tableau est référencée en utilisant le nom du tableau suivi de l'indice entre crochets**
- **Par exemple, l'expression**

```
scores[2]
```

réfère à la valeur 94 (la 3ième valeur du tableau)

- **Cette expression représente un endroit pour stocker un entier simple, et peut être utilisée partout où une variable entière peut l'être.**

- **Par exemple, nous pouvons lui assigner une valeur, imprimée, ou utilisée dans un calcul**

Tableaux

- **Un tableau stocke de multiples valeurs d'un même type**
- **Ce type peut être un type de base ou un objet**
- **Ainsi, nous pouvons créer un tableau d'entiers, ou un tableau de caractères, ou un tableau d'objets String, etc.**
- **En Java, le tableau lui-même est un objet**
- **Donc le nom du tableau est une variable de référence à un objet, et le tableau lui-même est instancié séparément**

Déclarations de Tableaux

- **Le tableau `scores` peut être déclaré comme suit:**

```
int[] scores = new int[10];
```

- **Notez que le type du tableau ne spécifie pas sa taille, mais chaque objet de ce type a une taille spécifique**
- **Le type de la variable `scores` est `int[]` (un tableau d'entiers)**
- **Il est appliqué à un nouvel objet tableau qui peut contenir 10 entiers**
- **Voir `BasicArray.java` (page 270)**

Déclarations de Tableaux

- **Des exemples de déclarations de tableaux :**

```
float[] prices = new float[500];

boolean[] flags;
flags = new boolean[20];

char[] codes = new char[1750];
```

Vérification des Bornes

- **Une fois qu'un tableau est créé, il a une taille fixe**
- **Un indice utilisé dans un référence à un tableau doit spécifié un élément valide**
- **C'est à dire, une valeur d'indice doit être bornées (0 à N-1)**
- **L'interpréteur Java va lancer une exception si un indice de tableau est hors bornes**
- **Ceci est la *vérification automatique des bornes***

Vérification des Bornes

- **Par exemple, si le tableau `codes` peut contenir 100 valeurs, il ne peut être indicé qu'avec les nombres de 0 à 99**
- **Si `count` a la valeur 100, alors la référence suivante causera une `ArrayOutOfBoundsException`:**

```
System.out.println (codes[count]);
```

- **Il est fréquent d'introduire une *erreur de débordement de un* lors de l'utilisation de tableaux**

problème

```
for (int index=0; index <= 100; index++)
   codes[index] = index*50 + epsilon;
```

Vérification des Bornes

- Chaque objet tableau possède une constante publique nommée `length` qui stocke la taille du tableau
- Elle est référencée en se servant du nom du tableau (comme pour tout autre objet):

```
scores.length
```

- Remarque : `length` contient le nombre d'éléments, et non l'indice le plus grand
- Voir `ReverseNumbers.java` (page 272)
- Voir `LetterCount.java` (page 274)

Déclarations de tableaux

- **Les crochets du type tableau peuvent être associées avec le type d'élément ou avec le nom du tableau**
- **Donc les déclarations suivantes sont équivalentes:**

```
float[] prices;
float prices[];
```

- **Le premier format est généralement plus lisible**

Listes d'initialisations

- **Une *liste* d'*initialisation* peut être utilisée pour instancier et initialiser un tableau en une seule étape**
- **Les valeurs sont délimitées par des accolades et séparées par des virgules**
- **Exemples:**

```
int[] units = {147, 323, 89, 933, 540,
               269, 97, 114, 298, 476};

char[] letterGrades = {'A', 'B', 'C', 'D', 'F'};
```

Listes d'initialisations

- **Notez que lorsque une liste d'initialisation est utilisée :**
  - **L'opérateur `new` n'est pas utilisé**
  - **La taille n'est pas spécifiée**
- **La taille du tableau est déterminée par le nombre d'items dans la liste d'initialisation**
- **Une liste d'initialisation ne peut être utilisée qu'à la déclaration d'un tableau**
- **Voir `Primes.java` (page 278)**

Tableaux en Paramètres

- **Un tableau entier peut être passé en paramètre à une méthode**
- **Comme tout autre objet, la référence au tableau est passée, rendant les paramètres formel et actuel des alias l'un de l'autre**
- **Changer l'élément d'un tableau dans une méthode change l'original**
- **Un élément d'un tableau peut aussi être passé à une méthode, les règles de passage de paramètres pour ce type d'éléments devra être suivit**

Tableaux d'Objets

- Les éléments d'un tableau peuvent être des références à des objets
- La déclaration suivante réserve de l'espace pour stocker 25 références à des objets `String`

```
String[] words = new String[25];
```

- Elle ne crée PAS les objets `String` eux mêmes
- Chaque objet stocké dans un tableau doit être instancié séparément
- Voir `GradeRange.java` (page 280)

Arguments en Ligne

- La signature de la méthode `main` indique qu'elle prend un tableau d'objets `String` en paramètre
- Ces valeurs proviennent des arguments en ligne qui sont fournis lorsque l'interpréteur est invoqué
- Par exemple, l'invocation de l'interpréteur suivante passe un tableau de trois objets `String` à la méthode main:

```
> java DoIt pennsylvania texas california
```

- Ces Strings sont stockée aux indices 0-2 des paramètres
- Voir `NameTag.java` (page 281)

Tableaux d'Objets

- **Les objets peuvent avoir des tableaux comme variables d'instance**
- **Donc, des structures assez complexes peuvent être créées simplement avec des tableaux et des objets**
- **Le concepteur de logiciel doit déterminer avec précaution une organisation des données et des objets qui est appropriée à la situation**
- **Voir `Tunes.java` (page 282)**
- **Voir `CDCollection.java` (page 284)**
- **Voir `CD.java` (page 286)**

Tri

- **Le tri est un processus qui consiste à placer les items d'une liste dans un ordre particulier**
- **Il doit y avoir une valeur sur laquelle l'ordre est basée**
- **Il existe plusieurs algorithmes de tri pour une liste d'items**
- **Ces algorithmes varient au niveau de leur efficacité**
- **Nous allons voir deux algorithmes spécifiques :**
  - **Tri par sélection**
  - **Tri par insertion**

Tri par Sélection

- **L'approche du Tri par Sélection :**
  - **sélectionnons une valeur et plaçons la à sa position finale dans la liste triée**
  - **répétons pour les autres valeurs**
- **Plus en détails :**
  - **Trouvons la plus petite valeur de la liste**
  - **Échangeons sa position avec la valeur à la première position**
  - **Trouvons la plus petite valeur suivante de la liste**
  - **Échangeons sa position avec la valeur à la seconde position**
  - **répétons jusqu'à ce que toutes les valeurs soient placées**

Tri par Sélection

- **Un exemple:**

```
original:              3   9   6   1   2
plus petit est 1:      1   9   6   3   2
plus petit est 2:      1   2   6   3   9
plus petit est 3:      1   2   3   6   9
plus petit est 6:      1   2   3   6   9
```

- **Voir `SortGrades.java` (page 289)**
- **Voir `Sorts.java` (page 290) -- la méthode `selectionSort`**

Tri par Insertion

- **L'approche du Tri par Insertion :**
  - **Choisir un item et l'insérer à la bonne position dans une sous-liste triée**
  - **répéter jusqu'à ce que tous les items aient été insérés**
- **Plus en détails:**
  - **considérons le premier item comme étant la sous-liste triée (d'un seul item)**
  - **insérons le second item dans la sous-liste triée, tout en déplaçant les au besoin pour créer l'espace nécessaire à l'insertion d'un nouvel élément**
  - **insérons le troisième item dan la sous-liste triée (de 2 items), en déplaçant les éléments au besoin**
  - **répétons jusqu'à ce que toutes les valeurs soient insérées à leur position**

Tri par Insertion

- **Un exemple:**

```
original:        3   9   6   1   2
insérons 9:      3   9   6   1   2
insérons 6:      3   6   9   1   2
insérons 1:      1   3   6   9   2
insérons 2:      1   2   3   6   9
```

- **Voir `Sorts.java` (page 290) – la méthode `insertionSort`**

Trier des objets

- Les entiers ont un ordre inhérent , mais l'ordre d'un ensemble d'objets doit être définie par la personne qui définie la classe
- Rappelez-vous qu'une interface Java peut être utilisée en tant que nom de type et garantit qu'une classe particulière a implémentée des méthodes particulières
- Nous pouvons utiliser l'interface `Comparable` pour développer un tri générique pour un ensemble d'objets
- Voir `SortPhoneList.java` (page 294)
- Voir `Contact.java` (page 295)
- Voir `Sorts.java` (page 290)

Comparaison des tris

- **Les tris par Sélection et par Insertion ont une efficacité semblable**
- **Ils ont chacun une boucle externe qui parcourt tous les éléments, et une boucle interne qui compare les valeurs de la boucle externe avec presque toutes les valeurs de la liste**
- **Donc approximativement n^2 comparaisons sont nécessaires pour trié une liste de taille n**
- **Nous affirmons que ces tris sont de *l'ordre de* n^2**
- **D'autres tris sont plus efficace: *ordre de* $n \log_2 n$**

Tableaux à deux-Dimensions

- **Un *tableau à une-dimension* stock une liste simple de valeurs**
- **Un *tableau à deux-dimensions* peut être vu comme une table de valeurs, avec des lignes et des colonnes**
- **Une référence à un élément d'un tableau à deux-dimensions en utilisant deux valeurs d'indices**
- **Pour être précis, un tableau à deux-dimensions en Java est un tableau de tableaux**
- **Voir `TwoDtableau.java` (page 299)**

Tableaux Multidimensionnels

- **Un tableau peut avoir autant de dimensions que nécessaires, créant ainsi un tableau multidimensionnel**
- **Chaque dimension subdivise la précédente en un nombre spécifique d'éléments**
- **Chaque dimension du tableau a sa propre constante `length`**
- **Comme chaque dimension est un tableau de références à des tableaux, les tableaux à l'intérieur d'une dimension peuvent être de différentes longueurs**

La Classe `vector`

- **Un objet de la classe `vector` est similaire à un tableau car il stock plusieurs valeurs**
- **Par contre, un vecteur**
  - **ne stock que des objets**
  - **Ne requiert pas la syntaxe d'indices comme les tableaux**
- **Les méthodes de la classe `vector` sont utilisées pour interagir avec les éléments d'un vecteur**
- **La classe `vector` fait partie du package `java.util`**
- **Voir `Beatles.java` (page 304)**

La Classe `vector`

- **Une différence importante entre un tableau et un vecteur est qu'un vecteur peut être vu comme étant dynamique, capable de changer sa taille si nécessaire**
- **Chaque vecteur a une certaine quantité de mémoires qui lui est initialement réservé pour stocker des éléments**
- **Si un élément est ajouté alors que espace disponible n'est pas suffisante, plus d'espace est automatiquement allouée**

La Classe `vector`

- La classe `vector` est implémentée en utilisant un tableau
- Lorsque de l'espace additionnelle est requise, un nouveau, et plus grand tableau est créé, et les valeurs y sont copiées à partir du tableau original
- Pour insérer un élément, les éléments existant sont d'abord copiées, un à un, vers une autre position dans le tableau
- Donc, l'implémentation de `vector` dans l'API n'est pas très efficace pour insérer des éléments

Polygones et Polylignes

- **Les tableaux sont souvent utiles dans le traitement de graphiques**
- **Polygones et polylignes sont des formes qui sont définit par des valeurs stockées dans des tableaux**
- **Un polyligne est semblable à un polygone sauf que ses lignes ne se rencontrent pas, donc il ne peut être rempli**
- **Voir `Rocket.java` (page 307)**
- **Il y a aussi une classe `Polygon` qui peut être utilisées pour définir et dessiner des polygones**

Sauvegarder l'état des Dessins

- **Chaque fois que la méthode `repaint` est appelée pour un applet, la fenêtre est vidée avant l'appel à `paint`**
- **Un tableau ou un vecteur peut être utilisé pour stocker l'objet dessiné, et le redessiné chaque fois que nécessaire**
- **Voir `Dots2.java` (page 310)**