Arrays

Chapter 7
Array Definitions

An array is a structured data type
- Contains many variables of the same data type (called elements)
- Allows manipulation of all variables with one name
- Individual elements accessed by an index
- Legal index values are 0..length - 1 (zero-based indexing)
  - Index bounds are checked

```java
<element type>[ ] <name> = new <element type>[ <size> ];
- int[ ] scores = new int[10]
```
Arrays

Always are instantiated objects

- Reference definition is distinct from object instantiation
  - `int[ ] numberOfDays;` // reference variable definition
  - `numberOfDays = new int[12];` // object instantiation
  - `int[ ] numberOfDays = new int[12];` // both

- Static instantiation and initialization
  - `int[ ] numberOfDays = { 31,28,31,30,31,30,31,31,30,31,30,31 };`

- `numberOfDays.length;` // capacity, not filled slots

- Bracket placement
  - `int[] numberOfDays;` // preferred
  - `int numberOfDays[];`
  - Useful for method arguments: `void func(int[ ] scores) {...}
  - Useful for method return types: `int[ ] func( ) {...}`
One-dimensional Array

```java
int[] test = new int[10];
```

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>41</td>
</tr>
<tr>
<td>1</td>
<td>97</td>
</tr>
<tr>
<td>2</td>
<td>91</td>
</tr>
<tr>
<td>3</td>
<td>89</td>
</tr>
<tr>
<td>4</td>
<td>89</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
</tr>
<tr>
<td>6</td>
<td>76</td>
</tr>
<tr>
<td>7</td>
<td>83</td>
</tr>
<tr>
<td>8</td>
<td>91</td>
</tr>
<tr>
<td>9</td>
<td>79</td>
</tr>
</tbody>
</table>

- `test[0]` is 41
- `test[5]` is 100
Two-dimensional Array

float[][] testScore = new float[10][4];

testScore[0][0] = 41;
testScore[2][3] = 91;
testScore[3][2] = 89;
testScore[5][0] = 100;
testScore[6][1] = 76;
testScore[7][2] = 83;
testScore[8][1] = 91;
testScore[9][3] = 79;
Three-dimensional Array

double[][][] classScore = new double[5][5][5];

classScore[0][2][3]
classScore[0][0][0]
classScore[2][1][0]
classScore[2][4][1]
classScore[4][3][0]
Array Syntax

Using arrays

- Arrays are often used with for-loops
- One loop per dimension
- Index goes from 0 to array.length - 1
- Examples
  ```java
  for (int i = 0; i < test.length; i++)
    System.out.println(test[i]);
  ```
  ```java
  for (int i = 0; i < testScore.length; i++)
    for (int j = 0; j < testScore[i].length; j++)
      System.out.println(testScore[i][j]);
  ```
The For-Each Loop

Adapted from C#

- for (<type> <name> : <array>) { . . . }
- for (int i : scores)
  System.out.println(i)
- for (String s : args)
  System.out.println(s);
Arrays of Objects

Arrays of objects are really arrays of references to objects

```java
Employee[ ] emp;
emp = new Employee[5];
for (int i = 0; i < emp.length; i++)
  emp[i] = new Employee();
```
Multidimensional Arrays

An array of references

- Array of references to arrays
- First subscript \((i)\) indexes array of references (i.e., rows)
- Second subscript \((j)\) indexes an element in the row
- Rows may be different lengths (each row is instantiated separately)
Command-Line Arguments

Array of Strings (see p. 414)

```java
public static void main (String[] args)
```

```
c:\> java ArgDemo hello world from CS1400
```

```
for (i = 0; i < args.length; i++)
    process(args[i]);
```

```
for (String s : args)
    process(s);
```
### Example-- Matrix Addition

Arrays on either side of the assignment operator

\[
A = \begin{bmatrix}
  a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\
  a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} \\
  \vdots & \vdots & \ddots & \vdots \\
  a_{m-1,0} & a_{m-1,1} & \cdots & a_{m-1,n-1}
\end{bmatrix}
\]

\[
B = \begin{bmatrix}
  b_{0,0} & b_{0,1} & \cdots & b_{0,n-1} \\
  b_{1,0} & b_{1,1} & \cdots & b_{1,n-1} \\
  \vdots & \vdots & \ddots & \vdots \\
  b_{m-1,0} & b_{m-1,1} & \cdots & b_{m-1,n-1}
\end{bmatrix}
\]

\[
C = A + B \rightarrow c_{i,j} = a_{i,j} + b_{i,j}
\]

```java
double[][][]a = new double[m][n];
double[][][]b = new double[m][n];
double[][][]c = new double[m][n];
for (int i = 0; i < m; i++)
  for (int j = 0; j < n; j++)
    c[i][j] = a[i][j] + b[i][j];
```
Example-- Matrix Multiplication

Arrays and loops: frequent companions

\[ C = AB \rightarrow c_{i,j} = \sum_{k=0}^{p-1} a_{i,k} b_{k,j} \]

double[][] a = new double[m][p];
double[][] b = new double[p][n];
double[][] c = new double[m][n];

for (int i = 0; i < m; i++)
    for (int j = 0; j < n; j++)
        { c[i][j] = 0;
        for (int k = 0; k < p; k++)
            c[i][j] += a[i][k] * b[k][j];
        }
Sorting

java.util.Arrays

- static void sort(type[ ] a)
- static void sort(type[ ] a, int fromIndex, int toIndex)
  ▶ fromIndex - the index of the first element (inclusive) to be sorted
  ▶ toIndex - the index of the last element (exclusive) to be sorted
- *type can be any built-in type*
  ▶ The sorting algorithm is a tuned quicksort
  ▶ This algorithm offers n*\log(n) performance
- *type can be Object*
  ▶ The sorting algorithm is a modified mergesort
  ▶ This sort is guaranteed to be *stable*: equal elements will not be reordered as a result of the sort
  ▶ guaranteed n*\log(n) performance
  ▶ All elements in the array must implement the **Comparable** interface
Searching

java.util.Arrays

- static int binarySearch(type[] a, type key)
  - The array must be sorted into ascending order according to the natural ordering of its elements
  - If the array contains multiple elements with the specified value, there is no guarantee which one will be found
  - Returns the index if key is found, otherwise a negative value
  - type may be any built-in type
  - type may a class type
    - Class must implement Comparable interface from java.lang

```java
interface Comparable<type>
{
    public int compareTo(type o);
}
```