Chapter 5: Arrays & Array Lists

Java™ How to Program, 11th Edition

Instructor: Zhuozhao Li

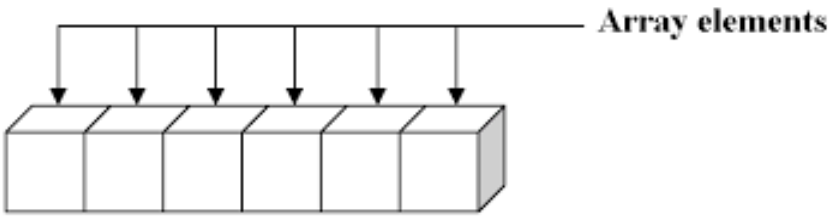
© 1992-2010 by Pearson Education, Inc. All Rights Reserved.

Objectives

- Arrays （数组）
- Use arrays to store data in and retrieve data from lists and tables of values
- Declare arrays, initialize arrays and refer to individual elements of arrays
- Use the enhanced `for` statement to iterate through arrays
- Declare and manipulate multidimensional arrays

©1992-2010 by Pearson Education, Inc. All Rights Reserved.

Arrays



- **Data structure （数据结构）:** collections of related data item
  - a data organization, management, and storage format that enables efficient access and modification
- An **array** (a widely-used data structure) is a group of variables (elements) containing values of the same type.
- Arrays are objects, so they're considered reference types.
- Elements can be either primitive types or reference types.

©1992-2010 by Pearson Education, Inc.
All Rights Reserved.

Primitive Types. vs. Reference Types

- Java types have two categories: primitive types 原始类型 and reference types 引用类型.
- Primitive types are the basic types of data
  - `byte, short, int, long, float, double, boolean, char`
  - A primitive-type variable can store one value of its declared type
- All non-primitive types are reference types. Programs use reference-type variables to store the locations of objects in memory.
- A reference-type variable is said to refer to an object in the program. Objects that are referenced may each contain many instance variables of primitive and reference types.

©1992-2010 by Pearson Education, Inc. All Rights Reserved.

Referring to Array Elements

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1

array-access expression

c[5] refers to the 6th element

c is the name of the reference to the array (name of the array for short)

5 is the position number of the element (index 索引 or subscript 下标)

Why does index start from 0?

https://www.geeksforgeeks.org/why-array-index-starts-from-zero/

©1992-2010 by Pearson Education, Inc.
All Rights Reserved.

Referring to Array Elements

- No space between the array name and the square brackets (`[ ]`), spaces after `[` or before `]` are allowed (`c[  8  ]`)
- The first element in every array has index zero
- An index must be a nonnegative integer
- A program can use an expression as an index (`c[ 1 + a ]`)
- The highest index in an array is *the number of elements* - 1
- Array names follow the same conventions as other variable names (Lower Camel Case https://en.wikipedia.org/wiki/Camel_case)
- Array-access expressions can be used on the left side of an assignment to place a new value into an array element (`c[1] = 2`)

©1992-2010 by Pearson Education, Inc.
All Rights Reserved.

Array Length

- Every array object knows its own length and stores it in a `length` instance variable (`c.length`)
- Even though the `length` instance variable of an array is `public`, it cannot be changed because it's a `final` variable (the keyword `final` creates constants).
- `c.length = 5 , wrong!!!`

©1992-2010 by Pearson Education, Inc.
All Rights Reserved.

Declaring (声明) and Creating Arrays

- Like other objects, arrays are created with keyword `new`.
- To create an array object, you specify the type of the array elements and the number of elements as part of an array-creation expression that uses keyword `new`.
  - `int[] c = new int[ 12 ];`
  - Returns a reference (representing the memory address of the array) that can be stored in an array variable

©1992-2010 by Pearson Education, Inc.
All Rights Reserved.

Declaring and Creating Arrays

```
int[] c = new int[ 12 ];
```

- The square brackets following the type indicate that the variable will refer to an array

- When type of the array and the square brackets are combined at the beginning of the declaration, all the identifiers in the declaration are array variables.

```
int[] a, b= new int[10];
System.out.println(b.length);
```

©1992-2010 by Pearson Education, Inc.
All Rights Reserved.

Declaring and Creating Arrays

- A program can declare arrays of any type.
- Every element of a primitive-type array contains a value of the array's declared element type.
  - `int[] c = new int[ 12 ];`
- Similarly, in an array of a reference type, every element is a reference to an object of the array's declared element type.
  - `Student[] students = new Student[ 12 ];`

©1992-2010 by Pearson Education, Inc.
All Rights Reserved.

Initializing the elements of an array to default values of zero

```
public class InitArray {
    public static void main(String[] args) {
        int[] array;  // declare an array

        array = new int[10];  // this creates the array object

        System.out.printf( "%s%8s\n", "Index", "Value" );

        // output each array element's value
        for (int counter = 0; counter < array.length; counter ++){
        System.out.printf("%5d%8d\n", counter, array[counter]);
        }
    } // end method main
}
```

Array Initialization

- You can create an array and initialize its elements with an array initializer—a comma-separated list of expressions (initializer list) enclosed in braces.

- `int[] n = { 10, 20, 30, 40, 50 };`
  - When the compiler sees an array declaration that includes an initializer list, it counts the number of initializers in the list to determine the size of the array, then sets up the appropriate `new` operation "behind the scenes".
  - Element `n[0]` is initialized to `10`, `n[1]` is initialized to `20`, and so on.

©1992-2010 by Pearson Education, Inc.
All Rights Reserved.

Elements can also be initialized one by one

```
public class InitArray1 {
   public static void main(String[] args) {
      int[] array;  // declare an array
      array = new int[10];  // this creates the array object

      // calculate the values of for each array element
      for (int counter = 0; counter < array.length; counter ++){
         array[counter] =  2 + 2 * counter;
      }

      System.out.printf( "%s%8s\n", "Index", "Value" );
      // output each array element's value
      for (int counter = 0; counter < array.length; counter ++){
      System.out.printf("%5d%8d\n", counter, array[counter]);
      }
   } // end method main
}
```

©1992-2010 by Pearson Education, Inc.
All Rights Reserved.

Array elements often represent a series of values to be used in a calculation

```
public class SumArray {
    public static void main(String[] args) {
        int[] array = {85, 96, 95, 83, 78, 99, 100};  // declare an array
        int total = 0;

        // add each element's value to total
        for (int counter = 0; counter < array.length; counter ++){
            total += array[counter];
        }
        System.out.printf("The sum of the array elements is %d\n", total);
    } // end method main
}
```

```
The sum of the array elements is 636
```

©1992-2010 by Pearson Education, Inc.
All Rights Reserved.

A Die-Rolling Program (擲骰子)

- We can use separate counters in a die-rolling program to track the number of occurrences of each side of a six-sided die as the program rolls the die 6000 times.
  - `int faceOneFreq, faceTwoFreq, …`

- Why not use an array?

©1992-2010 by Pearson Education, Inc.
All Rights Reserved.

```java
import java.util.Random;

public class RollDie {
   public static void main(String[] args) {
      Random randomNumberGenerator = new Random();  // random
number generator
      int[] frequency = new int[7];
      // roll 6000 times; use dice value as frequency index
      for (int roll = 0; roll < 6000; roll ++){
         frequency[1 + randomNumberGenerator.nextInt(6)]++;
      }
      System.out.printf("%s%10s\n", "Value", "Frequency");
      // output the frequency of each value
      for(int value = 1; value < frequency.length; value ++){
         System.out.printf("%4d%10d\n", value, frequency[value]);
      }
   } // end method main
}
```

©1992-2010 by Pearson Education, Inc.
All Rights Reserved.

Array Bound Checking

- The JVM checks array indices to ensure that they're greater than or equal to 0 and less than the length of the array
- If a program uses an invalid index, JVM throws an exception to indicate that an error occurred at runtime.
  - Invalid indices often occur when accessing array elements in loops

```
int[] a = new int[10];
for(int i = 0; i <= 10; i++) a[i] = i;
```

```
java.lang.ArrayIndexOutOfBoundsException: 10
```

©1992-2010 by Pearson Education, Inc.
All Rights Reserved.

Enhanced for Statement

```
for ( parameter : arrayName ) {
   statement(s)
}
```

```
for ( int num : numbers ) {
   total += num;
}
```

- Iterates through the elements of an array without using a counter, thus avoiding the possibility of "stepping outside" the array.
  - *parameter* has a type and an identifier
  - *arrayName* is the array through which to iterate.
  - Parameter type must be consistent with the type of the elements in the array.

©1992-2010 by Pearson Education, Inc.
All Rights Reserved.

Enhanced for Statement

- Simple syntax compared to the normal `for` statement

```
for ( int num : numbers ) {
   // statements using num
}
```

Semantically equivalent to:

```
for ( int i = 0; i < numbers.length; i++ ) {
   int num = numbers[i];
   // statements using num
}
```

©1992-2010 by Pearson Education, Inc.
All Rights Reserved.

Enhanced for Statement

- Often used to replace counter-controlled `for` statement when the code requires access only to element values.

```
for ( int i = 0; i < numbers.length; i++ ) {
   total += numbers[i];
}
```

Simpler and elegant ☺

```
for ( int num : numbers ) {
   total += num;
}
```

©1992-2010 by Pearson Education, Inc.
All Rights Reserved.

Enhanced for Statement

- Cannot be used to modify element values

```
for ( int num : numbers ) {
   num = 0;
}
```

Can this change the array element values?

No! Only change the value of `num`

Semantically equivalent to:

```
for ( int i = 0; i < numbers.length; i++ ) {
   int num = numbers[i];
   num = 0;
}
```

Local variable `num` stores a copy of the array element value

©1992-2010 by Pearson Education, Inc.
All Rights Reserved.

Two-Dimensional Arrays

- Arrays that we have considered up to now are one-dimensional arrays: a single line of elements.

	78	-9	520	0	14
Index	0	1	2	3	4

Example: an array of five random numbers

©1992-2010 by Pearson Education, Inc. All Rights Reserved.

Two-Dimensional Arrays

- Data in real life often come in the form of a table

	Test 1	Test 2	Test 3	Test 4	Test 5
Student 1	87	96	70	68	92
Student 2	85	75	83	81	52
Student 3	69	77	96	89	72
Student 4	78	79	82	85	83

Example:
a gradebook

The table can be represented using a two-dimensional array in Java

©1992-2010 by Pearson Education, Inc. All Rights Reserved.

Two-Dimensional (2D) Arrays

- 2D arrays are indexed by two subscripts: one for the row number, the other for the column number

	Test 1	Test 2	Test 3	Test 4	Test 5
Student 1	87	96	70	68	92
Student 2	85	75	83	81	52
Student 3	69	77	96	89	72
Student 4	78	79	82	85	83

column

row

```
gradbook[ 1 ][ 2 ]
```

(gradebook is the name of the array)

©1992-2010 by Pearson Education, Inc. All Rights Reserved.

2D Array Details (Similar to 1D Array)

- Similar to 1D array, each element in a 2D array should be of the same type: either primitive type or reference type
- Array access expression (subscripted variables) can be used just like a normal variable: `gradebook[1][2] = 77;`
- Array indices (subscripts) must be of type `int`, can be a literal, a variable, or an expression: `gradebook[1][j]`
- If an array element does not exist, JVM will throw exception `ArrayIndexOutOfBoundException`

©1992-2010 by Pearson Education, Inc. All Rights Reserved.

Declaring and Creating 2D Arrays

```
int[][] gradebook;
```

- Declares a variable that references a 2D array of `int`

```
gradebook = new int[50][6];
```

- Creates a 2D array **(50-by-6 array)** with **50 rows** (for 50 students) and **6 columns** (for 6 tests) and assign the reference to the new array to the variable `gradebook`

- Shortcut: `int[][] gradebook = new int[50][6];`

©1992-2010 by Pearson Education, Inc.
All Rights Reserved.

Array Initialization

- Similar to 1D array, we can create a 2D array and initialize its elements with nested array initializers as follows
  - `int[][] a = { { 1, 2 }, { 3, 4 } };`

- In 2D arrays, rows can have different lengths (ragged arrays)
  - `int[][] a = {{1, 2, 3, 4}, {5, 6}, {7, 8, 9}, {10}};`

1	2	3	4
5	6		
7	8	9	
10			

Row 1 Row 2 Row 3 Row 4

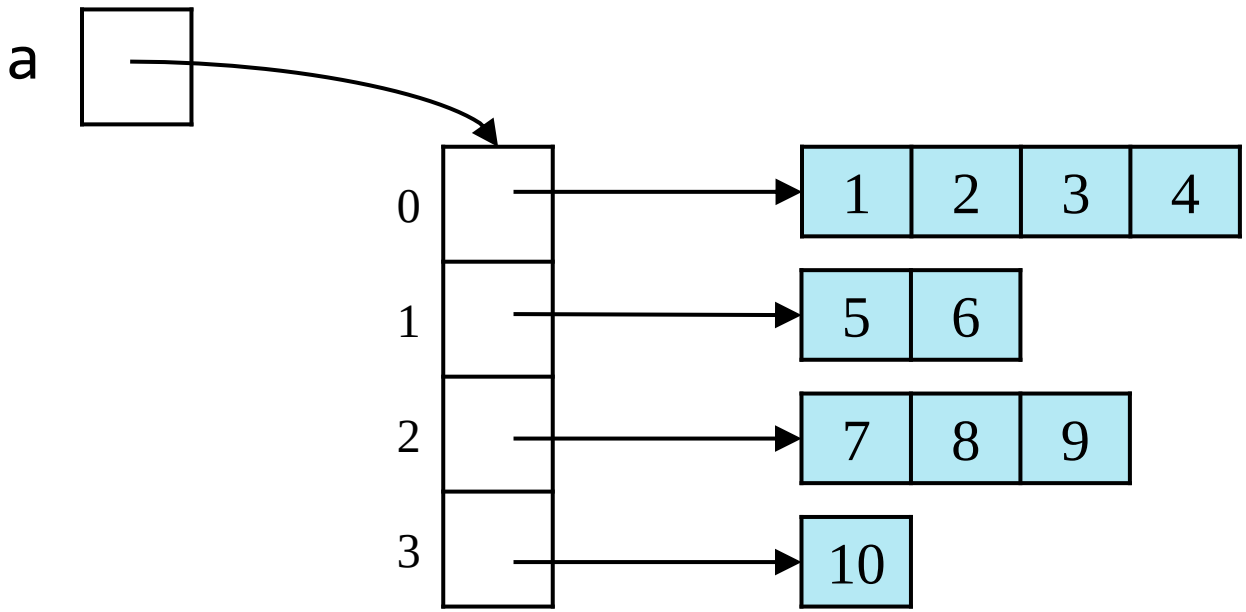
Note that the compiler will "smartly" determine the number of rows and columns

©1992-2010 by Pearson Education, Inc.
All Rights Reserved.

Under the Hood

- A 2D array is a 1D array of (references to) 1D arrays

```
int[][] a = {{1, 2, 3, 4}, {5, 6}, {7, 8, 9}, {10}};
```



©1992-2010 by Pearson Education, Inc. All Rights Reserved.

Under the Hood

```
int[][] a = {{1, 2, 3, 4}, {5, 6}, {7, 8, 9}, {10}};
```

- What is the value of `a[0]`?
  - Answer: The reference to the 1D array {1, 2, 3, 4}
- What is the value of `a.length`?
  - Answer: 4, the number of rows
- What the value of `a[1].length`?
  - Answer: 2, the second row only has 2 columns

©1992-2010 by Pearson Education, Inc. All Rights Reserved.

Declaring and Creating 2D Arrays

- Since a 2D array is a 1D array of (references to) 1D arrays, a 2D array in which each row has a different number of columns can also be created as follows:

```
int[][] b = new int[ 2 ][ ];   // create 2 rows
b[ 0 ] = new int[ 5 ]; // create 5 columns for row 0
b[ 1 ] = new int[ 3 ]; // create 3 columns for row 1
```

©1992-2010 by Pearson Education, Inc.
All Rights Reserved.

Displaying Element values

```
public static void main(String[] args) {
    int[][] a = {{1, 2, 3, 4}, {5, 6}, {7, 8, 9}, {10}};
    // loop through rows
    for(int row = 0; row < a.length; row++) {
        // loop through columns
        for(int column = 0; column < a[row].length; column++) {
            System.out.printf("%d ", a[row][column]);
        }
        System.out.println();
    }
}
```

```
1 2 3 4
5 6
7 8 9
10
```

©1992-2010 by Pearson Education, Inc. All Rights Reserved.

Computing Average Scores

```
public static void main(String[] args) {
    int[][] gradebook = {
        {87, 96, 70, 68, 92},
        {85, 75, 83, 81, 52},
        {69, 77, 96, 89, 72},
        {78, 79, 82, 85, 83}
    };
    for(int[] grades : gradebook) {
        int sum = 0;
        for(int grade : grades) {
            sum += grade;
        }
        System.out.printf("%.1f\n", ((double) sum)/grades.length);
    }
}
```

```
82.6
75.2
80.6
81.4
```

©1992-2010 by Pearson Education, Inc. All Rights Reserved.

Multidimensional Arrays

- Arrays can have more than two dimensions.
  - `int[][][] a = new int[3][4][5];`
- Concepts for multidimensional arrays (2D above) can be generalized from 2D arrays
  - 3D array is an 1D array of (references to) 2D arrays, which is a 1D array of (references to) 1D arrays
- 1D array and 2D arrays are most commonly-used.

©1992-2010 by Pearson Education, Inc. All Rights Reserved.

Swap the values of two elements

- `int[] n = { 10, 20 };`
- `n[0] = n[1]; n[1] = n[0];` ✗ ERROR
- `temp = n[1]; n[1] = n[0]; n[0] = temp;`

©1992-2010 by Pearson Education, Inc. All Rights Reserved.

6.11 Using Command-Line Arguments

- It's possible to pass arguments from the command line (these are known as command-line arguments) to an application by including a parameter of type `String[]` in the parameter list of `main`.
- By convention, this parameter is named `args`.
- When an application is executed using the `java` command, Java passes the command-line arguments that appear after the class name in the `java` command to the application's `main` method as `String`s in the array `args`.

©1992-2010 by Pearson Education, Inc.
All Rights Reserved.

```java
public class ArgsExample {
   public static void main(String[] args) {
      // check number of command-line arguments
      if (args.length != 3)
         System.out.printf(
            "Error: Please re-enter the entire command, including%n" +
                  "an array size, initial value and increment.%n");
      else{
         // get array size from first command-line argument
         int arrayLength = Integer.parseInt(args[0]);
         int[] array = new int[arrayLength];

         // get initial value and increment from command-line arguments
         int initialValue = Integer.parseInt(args[1]);
         int increment = Integer.parseInt(args[2]);
```

©1992-2010 by Pearson Education, Inc.
All Rights Reserved.

```
        // calculate value for each array element
        for (int counter = 0; counter < array.length; counter++)
            array[counter] = initialValue + increment * counter;

        System.out.printf("%s%8s%n", "Index", "Value");
         // display array index and value
        for (int counter = 0; counter < array.length; counter++)
            System.out.printf("%5d%8d%n", counter, array[counter]);
    }
  }
} // end class InitArray
```

©1992-2010 by Pearson Education, Inc.
All Rights Reserved.

6.12 Class `Arrays`

- Class Arrays helps you avoid reinventing the wheel by providing `static` methods for common array manipulations.
- These methods include sort for sorting an array (i.e., arranging elements into increasing order), binarySearch for searching an array (i.e., determining whether an array contains a specific value and, if so, where the value is located), equals for comparing arrays and fill for placing values into an array.
- These methods are overloaded for primitive-type arrays and for arrays of objects.
- You can copy arrays with class `System`'s `static` arraycopy method.
-

©1992-2010 by Pearson Education, Inc.
All Rights Reserved.

```java
// Fig. 6.16: ArrayManipulations.java
// Arrays class methods and System.arraycopy.
import java.util.Arrays;

public class ArrayManipulations
{
   public static void main( String[] args )
   {
      // sort doubleArray into ascending order
      double[] doubleArray = { 8.4, 9.3, 0.2, 7.9, 3.4 };
      Arrays.sort( doubleArray );
      System.out.printf( "\ndoubleArray: " );

      for ( double value : doubleArray )
         System.out.printf( "%.1f ", value );

      // fill 10-element array with 7s
      int[] filledIntArray = new int[ 10 ];
      Arrays.fill( filledIntArray, 7 );
      displayArray( filledIntArray, "filledIntArray" );

      // copy array intArray into array intArrayCopy
      int[] intArray = { 1, 2, 3, 4, 5, 6 };
      int[] intArrayCopy = new int[ intArray.length ];
```

Fig. 6.16 | Arrays class methods. (Part 1 of 4.)

©1992-2010 by Pearson Education, Inc.
All Rights Reserved.

```java
      System.arraycopy( intArray, 0, intArrayCopy, 0, intArray.length );
      displayArray( intArray, "intArray" );
      displayArray( intArrayCopy, "intArrayCopy" );

      // compare intArray and intArrayCopy for equality
      boolean b = Arrays.equals( intArray, intArrayCopy );
      System.out.printf( "\n\nintArray %s intArrayCopy\n",
         ( b ? "==" : "!=" ) );

      // compare intArray and filledIntArray for equality
      b = Arrays.equals( intArray, filledIntArray );
      System.out.printf( "intArray %s filledIntArray\n",
         ( b ? "==" : "!=" ) );

      // search intArray for the value 5
      int location = Arrays.binarySearch( intArray, 5 );

      if ( location >= 0 )
         System.out.printf(
            "Found 5 at element %d in intArray\n", location );
      else
         System.out.println( "5 not found in intArray" );

```

Fig. 6.16 | Arrays class methods. (Part 2 of 4.)

©1992-2010 by Pearson Education, Inc.
All Rights Reserved.

```java
      // search intArray for the value 8763
      location = Arrays.binarySearch( intArray, 8763 );

      if ( location >= 0 )
         System.out.printf(
            "Found 8763 at element %d in intArray\n", location );
      else
         System.out.println( "8763 not found in intArray" );
   } // end main

   // output values in each array
   public static void displayArray( int[] array, String description )
   {
      System.out.printf( "\n%s: ", description );

      for ( int value : array )
         System.out.printf( "%d ", value );
   } // end method displayArray
} // end class ArrayManipulations
```

Fig. 6.16 | `Arrays` class methods. (Part 3 of 4.)

©1992-2010 by Pearson Education, Inc.
All Rights Reserved.

```
doubleArray: 0.2 3.4 7.9 8.4 9.3
filledIntArray: 7 7 7 7 7 7 7 7 7 7
intArray: 1 2 3 4 5 6
intArrayCopy: 1 2 3 4 5 6

intArray == intArrayCopy
intArray != filledIntArray
Found 5 at element 4 in intArray
8763 not found in intArray
```

Fig. 6.16 | `Arrays` class methods. (Part 4 of 4.)

©1992-2010 by Pearson Education, Inc.
All Rights Reserved.

Assignment 1

- Due on Oct. 17, 2021
- Submit on OJ

©1992-2010 by Pearson Education, Inc. All Rights Reserved.