Arrays and ArrayLists

CS102A Lecture 5

James YU

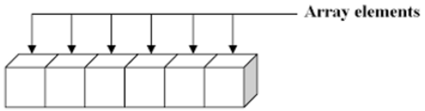
Oct. 12, 2020

Objectives

- What arrays are.
- 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.
- Read command-line arguments into a program.

Arrays

- Data structure: collections of related data item.
- Recall `String[]` args when you use command line arguments:
  - `args[0], args[1], ...`
- An array (a widely-used data structure) is a group of variables (elements) containing values of the same type.
- Elements can be either primitive types or reference types.

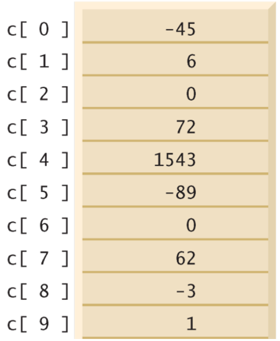


Primitive types. vs. reference types

- Java types have two categories: *primitive types* and *reference types*.
- 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.

Referring to array elements

- 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 or subscript).



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 and is sometimes called the zeroth element.
- 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.
- 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`).

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).

Declaring and creating arrays

- Arrays are created with keyword `new`.
- To create an array, you specify the type of the array elements and the number of elements as part of an array-creation expression that uses keyword `new`.

```java
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.

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

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.

```
GradeBook[] gradebooks = new GradeBook[12];
```

- Now we have several examples that demonstrate declaring arrays, creating arrays, initializing arrays and manipulating array elements.

Declaring and creating arrays

```java
// Initializing the elements of an array to default values of zero.
public class InitArray
{
  public static void main( String [] args ) {
    int[] array; // declare array named array
    array = new int[ 10 ]; // create the array object
    System.out.printf("%s%8s\n", "Index", "Value"); // column headings
    // output each array element's value
    for (int counter = 0; counter < array.length; counter++)
      System.out.printf("%5d%8d\n", counter, array[ counter ]);
  } // end main
} //end class InitArray
```

Declaring and creating arrays

```
Index    Value
    0        0
    1        0
    2        0
    3        0
    4        0
    5        0
    6        0
    7        0
    8        0
    9        0
```

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.

```java
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.

Array initialization

```java
// Initializing the elements of an array with an array initializer.
public class InitArray
{
  public static void main( String []] args ) {
    // initializer list specifies the value for each element
    int[] array = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37};

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

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

Array initialization

- Elements can also be initialized one by one.

```java
// Calculating values to be placed into elements of an array
public class InitArray
{
  public static void main( String []] args ) {
    final int ARRAY_LENGTH = 10; // declare constant
    int[] array = new int[ ARRAY_LENGTH ]; // create array

    for (int counter = 0; counter < array.length; counter++)
      array[counter] = 2 + 2 * counter
    System.out.printf("%s%8s\n", "Index", "Value"); // column headings

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

Array initialization

- Modifier `final` indicates that a variable is a constant.
- Constant variables must be initialized before they're used and cannot be modified thereafter.
- If you attempt to modify a final variable after it's initialized in its declaration, the compiler issues an error message like

```
cannot assign a value to final variable variableName
```

- If an attempt is made to access the value of a final variable before it's initialized, the compiler issues an error message like

```
variable variableName might not have been initialized
```

Examples using arrays

- Often, the elements of an array represent a series of values to be used in a calculation.

```java
// Computing the sum of the elements of an array.
public class SumArray
{
  public static void main( String[] args ) {
    int[] array = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
    int total = 0;

    // add each element's value to total
    for ( int counter = 0; counter < array.length; counter++ )
      total += array[ counter ];

    System.out.printf("Total of array elements: %d\n", total);
  } // end main
} //end class SumArray
```

Examples using arrays

- Many programs present data to users in a graphical manner.
- For example, numeric values are often displayed as bars in a bar chart.
- In such a chart, longer bars represent proportionally larger numeric values.
- One simple way to display numeric data graphically is with a bar chart that shows each numeric value as a bar of asterisks (*).

Examples using arrays

```java
//Bar chart printing program
public class BarChart
{
  public static void main( String[] args ) {
    int[] array = { 0, 0, 0, 0, 0, 0, 1, 2, 4, 2, 1};

    System.out.println("Grade distribution: ");

    // for each array element, output a bar of the chart
    for (int counter = 0; counter < array.length; counter++) {
      // output bar label ("00-09: ", ..., "90-99: ", "100: ")
      if (counter == 10)
        System.out.printf("%5d: ", 100 );
      else
        System.out.printf("%02d-%02d: ", counter * 10, counter * 10 + 9);
```

Examples using arrays

```
         // print bar of asterisks
         for ( int stars = 0; stars < array [counter]; stars++ )
           System.out.print( "*" );
         System.out.println(); // start a new line of output
      }
   }
}
```

Examples using arrays

```
Grade distribution:
00-09:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: *
70-79: **
80-89: ****
90-99: **
  100: *
```

A die-rolling program

- 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.

```java
int faceOneFreq, faceTwoFreq, ...
```

- Why not use an array?

```java
// Die-rolling program using arrays instead of switch
import java.util.Random;
public class RollDie
{
  public static void main( String[] args ) {
    Random randomNumbers = new Random(); // random number generator
    int[] frequency = new int [ 7 ]; // array of frequency counters
```

A die-rolling program

```
    // roll die 6000 times1 use die value as frequency index
    for ( int roll = 1; roll <= 6000; roll++ )
      ++frequency[ 1 + randomNumbers.nextInt( 6 )]

    System.out.printf( "%s%10s\n", "Face", "Frequency" );

    // output each array element's value
    for ( int face = 1; face < frequency.length; face++ )
      System.out.printf( "%04d%10d\n", face, frequency[ face ] );
  }
}
```

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
```

Enhanced `for` statement

```java
for (parameter : arrayName) {
  statements;
}
```

```java
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.

Enhanced `for` statement

- Simple syntax compared to the normal `for` statement:

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

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

  - They are equivalent.
- Can this change the array element values? No! Only change the value of `num`.
  - Local variable `num` stores a copy of the array element value.

Bubble sort

- Problem: Collect the scores from the students and print them in ascending order (from smallest number to largest).

```java
int[] score = {5 1 4 2 8};
```

- Bubble sort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent pairs and swaps them if they are in the wrong order.
  - In each step, elements written in bold are being compared.
- First pass:

  ❶ (**5 1** 4 2 8) → (**1 5** 4 2 8), swap since 5 > 1.
  ❷ (1 **5 4** 2 8) → (1 **4 5** 2 8), swap since 5 > 4.
  ❸ (1 4 **5 2** 8) → (1 4 **2 5** 8), swap since 5 > 2.
  ❹ (1 4 2 **5 8**) → (1 4 2 **5 8**), now since these elements are already in order (8 > 5), algorithm does not swap them.

Bubble sort

- Second pass:
  ❶ (**1** **4** 2 5 8) → (**1** **4** 2 5 8)
  ❷ (1 **4** **2** 5 8) → (1 **2** **4** 5 8), Swap since 4 > 2.
  ❸ (1 2 **4** **5** 8) → (1 2 **4** **5** 8)
  ❹ (1 2 4 **5** **8**) → (1 2 4 **5** **8**)
- How to you swap the value of two variables: `var1` and `var2`?

```
temp = var1;
var1 = var2;
var2 = temp;
```

Bubble sort

- The above example needs two pass to complete the sort, but the algorithm does not know.
- For an array of 5 elements, you need at most 4 passes to complete the sort with any input pattern.

```
for (int i = 0; i < score.length-1;
     i++)
  for (int j = 0; j < score.length
      -1; j++)
    if (score[j] > score[j+1]) {
      temp = score[j];
      score[j] = score[j+1];
      score[j+1] = temp;
    }
```

```
for (int i = 0; i < score.length-1;
     i++)
  for (int j = 0; j < score.length
      -1-i; j++)
    if (score[j] > score[j+1]) {
      temp = score[j];
      score[j] = score[j+1];
      score[j+1] = temp;
    }
```

Two-dimensional arrays

- Arrays that we have considered up to now are one-dimensional arrays: a single line of elements.
  - Example: an array of five random numbers:

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

Two-dimensional arrays

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

	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

- `gradebook[1][2] == 83`
  - `gradebook` is the name of the array.
  - `1` is the row number.
  - `2` is the column number.

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`.

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];`

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			

Under the hood

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

```java
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.

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:

```java
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
```

Displaying element values

```java
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
```

Computing average scores

```java
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);
  }
}
```

Multidimensional arrays

- Arrays can have more than two dimensions.

```java
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.

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`.

Using command-line arguments

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

Using command-line arguments

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

Using command-line arguments

```
> java InitArray
  Error: Please re-enter the entire command, including
  an array size, initial value and increment.
```

```
> java InitArray 5 0 4
  Index   Value
      0       0
      1       4
      2       8
      3      12
      4      16
```

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.
- You can copy arrays with class `System`'s static `arraycopy` method.

Class Arrays

```java
// Arrays class methods and System.arraycopy.
import java.util.Arrays;
public class ArrayManipulations
{
  // 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);
  }
  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: ");
```

Class `Arrays`

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

Class Arrays

```
    // compare intArray and intArrayCopy for equality
    boolean b = Arrays.equals(intArray, intArrayCopy);
    System.out.printf("\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");
```

Class Arrays

```
      // 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 class ArrayManipulations
```

Class `Arrays`

```
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
```