14. Array Basics

Java

Summer 2008

Instructor: Dr. Masoud Yaghini

Outline

- Introduction
- Declaring an Array
- Creating Arrays
- Accessing an Array
- Simple Processing on Arrays
- Copying Arrays
- References

Introduction

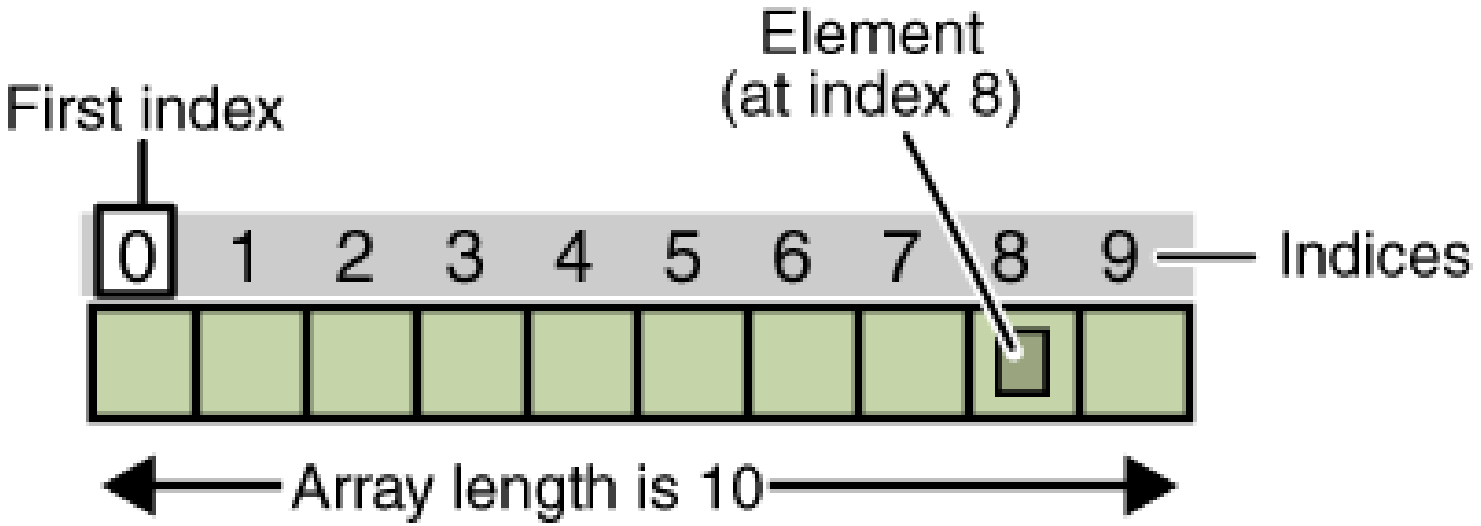
A problem with simple variables

- One variable holds one value
  - The value may change over time, but at any given time, a variable holds a single value
- If you want to keep track of many values, you need many variables
- All of these variables need to have names
- What if you need to keep track of hundreds or thousands of values?

Arrays

- An array lets you associate one name with a fixed number of values
- All values must have the same type
- Each item in an array is called an element
- The values are distinguished by a numerical index between 0 and array size minus 1
- The length of an array is established when the array is created.

An array of ten elements



An Example

```
class ArrayDemo {
    public static void main(String[] args) {
        int[] anArray; // declares an array of integers
        anArray = new int[10]; // allocates memory for 10 integers
        anArray[0] = 100; // initialize first element
        anArray[1] = 200; // initialize second element
        anArray[2] = 300; // etc.
        anArray[3] = 400;
        anArray[4] = 500;
        anArray[5] = 600;
        anArray[6] = 700;
        anArray[7] = 800;
        anArray[8] = 900;
        anArray[9] = 1000;

        System.out.println("Element at index 0: " + anArray[0]);
        System.out.println("Element at index 1: " + anArray[1]);
        System.out.println("Element at index 2: " + anArray[2]);
        System.out.println("Element at index 3: " + anArray[3]);
        System.out.println("Element at index 4: " + anArray[4]);
        System.out.println("Element at index 5: " + anArray[5]);
        System.out.println("Element at index 6: " + anArray[6]);
        System.out.println("Element at index 7: " + anArray[7]);
        System.out.println("Element at index 8: " + anArray[8]);
        System.out.println("Element at index 9: " + anArray[9]);
    }
}
```

The output from the program

```
Element at index 0: 100
Element at index 1: 200
Element at index 2: 300
Element at index 3: 400
Element at index 4: 500
Element at index 5: 600
Element at index 6: 700
Element at index 7: 800
Element at index 8: 900
Element at index 9: 1000
```

Declaring an Array

Declaring an Array

- Declaring an array:

```
datatype[] arrayRefVar;
```

- Example:

```
double[] myList;
```

- An array declaration has two components:
  - the array's type, and
  - the array's name

Declaring an Array

- An array's type is written as type[ ], where:
  - type is the data type of the contained elements;
  - the square brackets are special symbols indicating that this variable holds an array.
- The size of the array is not part of its type (which is why the brackets are empty).
- Unlike declarations for primitive data type variables, the declaration of an array variable does not allocate any space in memory for the array.

Declaring an Array

- An array's name can be anything you want, provided that it follows the rules and conventions as variables.
- The declaration does not actually create an array, it simply tells the compiler that this variable will hold an array of the specified type.

Declaring an Array

- Similarly, you can declare arrays of other types:
  - byte[ ] anArrayOfBytes;
  - short[ ] anArrayOfShorts;
  - long[ ] anArrayOfLongs;
  - float[ ] anArrayOfFloats;
  - double[ ] anArrayOfDoubles;
  - boolean[ ] anArrayOfBooleans;
  - char[ ] anArrayOfChars;
  - String[ ] anArrayOfStrings;

Declaring an Array

- You can also place the square brackets after the array's name:
  - float anArrayOfFloats[ ];
- However, convention discourages this form
- The brackets identify the array type and should appear with the type designation.

Declaring an Array

- You can declare more than one variable in the same declaration:

```
int a[ ], b, c[ ], d;   // notice position of brackets
```

  - a and c are int arrays
  - b and d are just ints

- Another syntax:

```
int [ ] a, b, c, d;      // notice position of brackets
```

  - a, b, c and d are int arrays

  – When the brackets come before the first variable, they apply to *all* variables in the list

- But, in Java, we typically declare each variable separately

Creating Arrays

Creating Arrays

- You cannot assign elements to an array unless it has already been created.
- After an array variable is declared, you can create an array by using the new operator with the following syntax:

```
arrayRefVar = new dataType[arraySize];
```

- This statement does two things:
  - (1) it creates an array using new dataType[arraySize];
  - (2) it assigns the reference of the newly created array to the variable arrayRefVar.

Creating Arrays

- Example:

  anArray = new int[10]; // create an array of integers

- This statement allocates an array with enough memory for ten integer elements and assigns the array to the anArray variable

- If this statement were missing, the compiler would print an error like the following, and compilation would fail:
  - Variable anArray might not have been initialized.

Declaring and Creating in One Step

- Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement, as shown below:

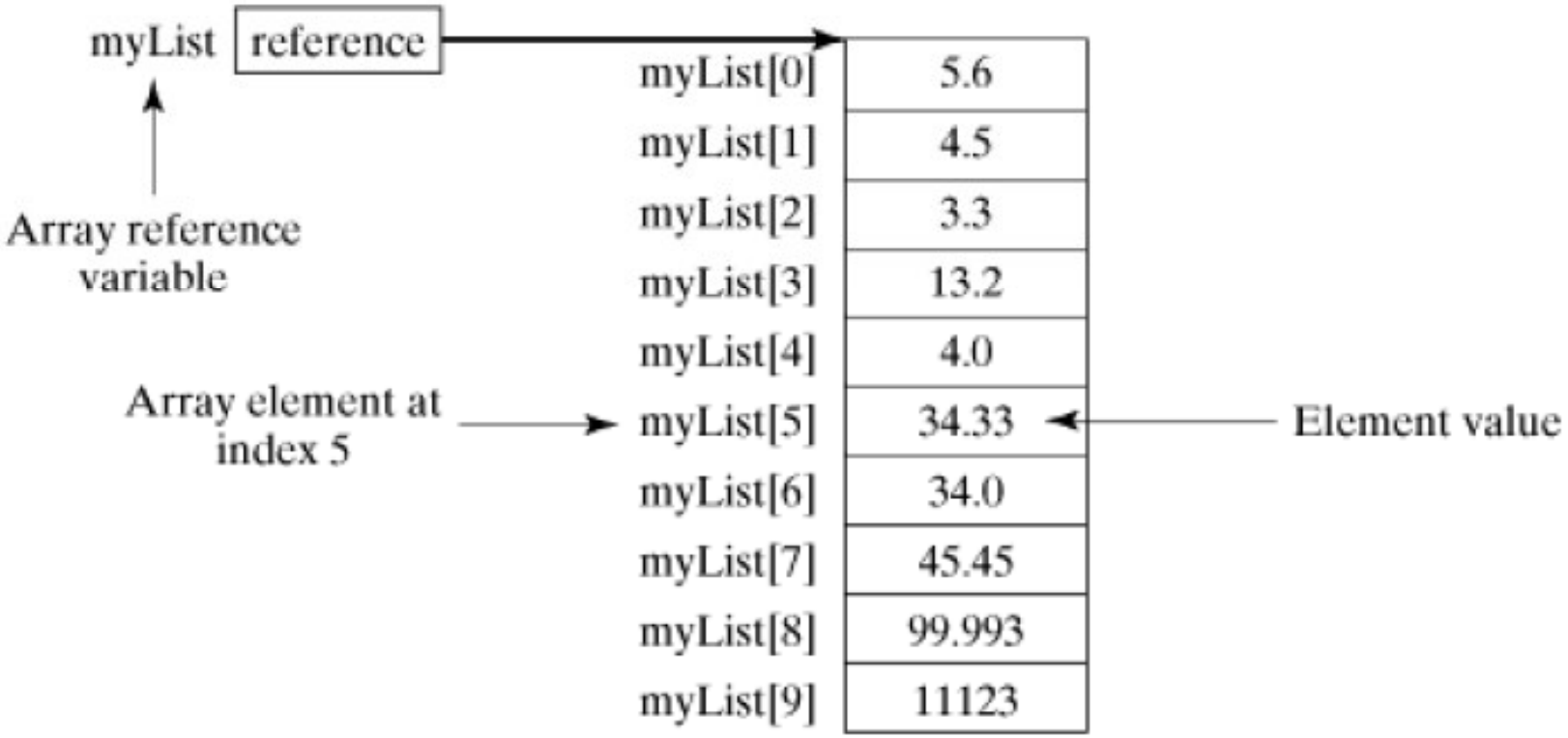
```
dataType[ ] arrayRefVar = new dataType[arraySize];
```

- Here is an example of such a statement:

```
double[ ] myList = new double[10];
```

Creating Arrays

```
double[] myList = new double[10];
```



Array Size

- Once an array is created, its size is fixed. It cannot be changed.

- you can use the built-in length property to determine the size of any array:
  - System.out.println(anArray.length);

- The code will print the array's size to standard output.

Default Values

- When an array is created, its elements are assigned the default value of:
  - 0 for the numeric primitive data types,
  - '\u0000' for char types, and
  - false for boolean types.

Accessing an Array

Accessing an Array

- The array elements are accessed through the index.
- The array indices are *0-based*, i.e., it starts from 0 to arrayRefVar.length-1.
- Each element in the array is represented using the following syntax, known as an *indexed variable*:

```
arrayRefVar[index];
```

Accessing an Array

- Each array element is accessed by its numerical index.
- Example:
  - System.out.println("Element 1 at index 0: " + anArray[0]);
  - System.out.println("Element 2 at index 1: " + anArray[1]);
  - System.out.println("Element 3 at index 2: " + anArray[2]);

Initializing an Array

- The next few lines assign values to each element of the array:
  - anArray[0] = 100; // initialize first element
  - anArray[1] = 200; // initialize second element
  - anArray[2] = 300; // initialize third element

Accessing an Array

	0	1	2	3	4	5	6	7	8	9
myArray	12	43	6	83	14	-57	109	12	0	6

- Examples:
  - x = myArray[1]; // sets x to 43
  - myArray[4] = 99; // replaces 14 with 99
  - m = 5;
    y = myArray[m]; // sets y to -57
  - z = myArray[myArray[9]]; // sets z to 109

Declare, Create and Initialize an Array

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
```

- This shorthand notation is equivalent to the following statements:

```
double[] myList = new double[4];

myList[0] = 1.9;

myList[1] = 2.9;

myList[2] = 3.4;

myList[3] = 3.5;
```

- Here the length of the array is determined by the number of values provided between { and }.

Caution

- Using the shorthand notation, you have to declare, create, and initialize the array all in one statement.
- Splitting it would cause a syntax error. For example, the following is wrong:

```
double[ ] myList;
myList = {1.9, 2.9, 3.4, 3.5};
```

An Example

```
1  public class Test {
2
3     public static void main(String[] args) {
4        int[] values = new int[5];
5
6        for (int i = 1; i < 5; i++) {
7           values[i] = i + values[i-1];
8        }
9        values[0] = values[1] + values[4];
10
11       for (int i = 0; i < values.length; i++) {
12         System.out.print(values[i] + " ");
13       }
14
15
16    }
17 }
```

Simple Processing on Arrays

Processing Arrays

- When processing array elements, you will often use a for loop.
- Here are the reasons why:
  - All of the elements in an array are of the same type. They are evenly processed in the same fashion by repeatedly using a loop.
  - Since the size of the array is known, it is natural to use a for loop.

Initializing arrays

- The following loop initializes the array myList with random values between 0.0 and 99.0:

```
for (int i = 0; i < myList.length; i++) {
   myList[i] = Math.random() * 100;
}
```

- Math.random() generates a random double value greater than or equal to 0.0 and less than 1.0 (0.0 <= Math.random() < 1.0).

Printing arrays

- To print an array:

```
for (int i = 0; i < myList.length; i++) {
    System.out.print(myList[i] + " ");
}
```

- For an array of the char[ ] type, it can be printed using one print statement.
- For example, the following code displays Dallas:

```
char[ ] city = {'D', 'a', 'l', 'l', 'a', 's'};
System.out.println(city);
```

Finding the largest element

- Use a variable named max to store the largest element:

```
double max = myList[0];
for (int i = 1; i < myList.length; i++) {
    if (myList[i] > max)
        max = myList[i];
}
```

Finding the smallest index of the largest element

- Often you need to locate the largest element in an array. If an array has more than one largest element, find the smallest index of such an element.

```
double max = myList[0];
int indexOfMax = 0;
for (int i = 1; i < myList.length; i++) {
    if (myList[i] > max) {
        max = myList[i];
        indexOfMax = i;
    }
}
```

Enhanced for statement

- Enhanced for statement can be used to make your loops more compact and easy to read.
- The following program uses the enhanced for to loop through the array:

```
class EnhancedForDemo {
    public static void main(String[] args) {
        int[] numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        for (int item : numbers) {
            System.out.println("Count is: " + item);
        }
    }
}
```

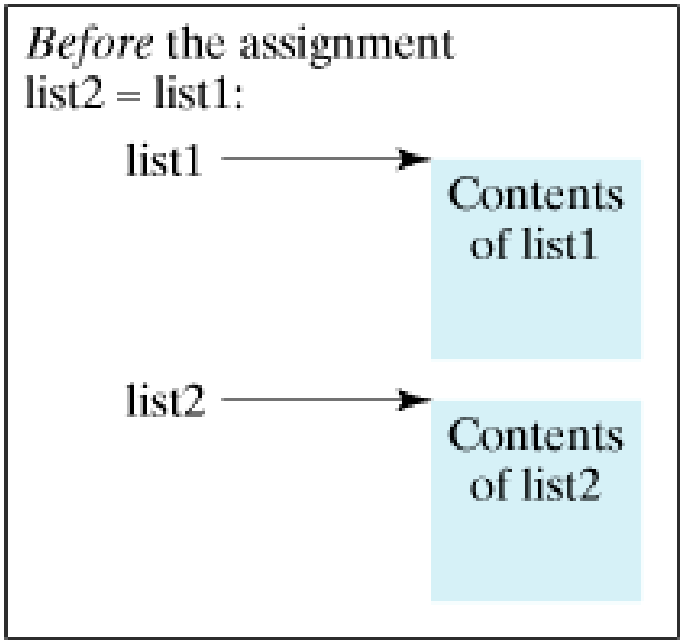
Copying Arrays

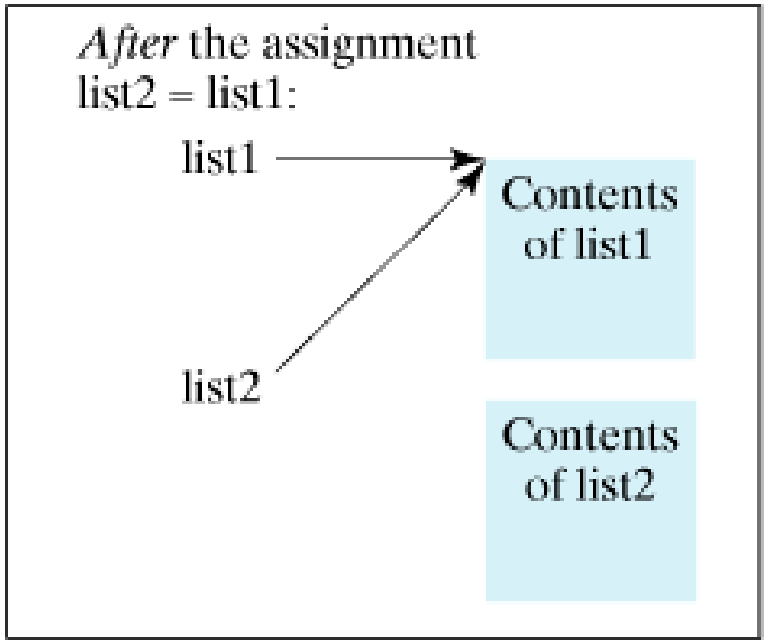
Copying Arrays

- Often you need to duplicate an array or a part of an array.
- In such cases you may attempt to use the assignment statement (=), as follows:

  list2 = list1;
- This statement does not copy the contents of the array referenced
- It merely copies the reference value from list1 to list2.
- The array previously referenced by list2 is no longer referenced; it becomes garbage

Before and after assignment





Copying Arrays

- Using a loop:

```
int[] sourceArray = {2, 3, 1, 5, 10};
int[] targetArray = new int[sourceArray.length];
for (int i = 0; i < sourceArray.length; i++) {
  targetArray[i] = sourceArray[i];
}
```

Copying Arrays

- The System class has an arraycopy method that you can use to efficiently copy data from one array into another:

```
arraycopy(sourceArray, srcPos,
  targetArray, tarPos, length);
```

- The arguments specify:
  - `sourceArray:` the array to copy from (source array)
  - `srcPos:` the array to copy to (destination array)
  - `targetArray:` the starting position in the source array
  - `tarPos:` the starting position in the destination array
  - `length:` the number of array elements to copy

ArrayCopyDemo Program

```
1  class ArrayCopyDemo {
2      public static void main(String[] args) {
3          char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e',
4                  'i', 'n', 'a', 't', 'e', 'd' };
5          char[] copyTo = new char[7];
6          System.arraycopy(copyFrom, 2, copyTo, 0, 7);
7          System.out.println(new String(copyTo));
8      }
9  }
```

- Output?

  caffein

ArrayCopyDemo Program

- The arraycopy method does not allocate memory space for the target array.
- The target array must have already been created with its memory space allocated.
- The arraycopy method violates the Java naming convention. By convention, this method should be named arrayCopy (i.e., with an uppercase C).

References

References

- Y. Daniel Liang, **Introduction to Java Programming**, Sixth Edition, Pearson Education, 2007. (Chapter 5)
- S. Zakhour and et. al., **The Java Tutorial: A Short Course on the Basics**, 4th Edition, Prentice Hall, 2006. (Chapter 3)

The End