

Arrays

Programming
Methodology and Design

Dr Ioanna Stamatopoulou

Arrays

- Declaration
- Initialisation
- Use of Arrays
- Arrays and Strings
- Arrays as method arguments

The need for Arrays

- The problem: we want to store 100 integers
 - Declaring 100 integer variables is not really practical...

```
int num1, num2, num3, ... , num100 ;
```

- Neither is initialising them and using them:

```
num1 = 0 ;  
...  
num100 = 0 ;
```

- Arrays solve this problem:

```
int num[] = new int[100];  
for (int i = 0 ; i < 100 ; i++)  
    num[i] = i ;  
  
for (int i = 0; i < num.length; i++)  
    System.out.println(num[i]);
```

Array Declaration

- To declare an array I declare its **type followed by a pair of square brackets** and its name:

```
String[] stringArray; //an array for Strings  
int[] integerArray; //an array for integers
```

- Declaring an array does not reserve memory space and does initialise its values
 - If you attempt to use an array immediately after declaring it you will get a compiler error: **Variable <array name> may not have been initialized**

Array Creation and Initialization

- To **create** an array you use the following syntax and also declare its **length** (size/number of positions):

```
stringArray = new String[10]; //an array for 10 Strings
integerArray = new int[5]; //an array for 5 integers
```

- **Declaration and creation** in one step:

```
char[] alphabet = new char[26];
double[] dnumbers = new double[10];
```

- Creation and **initialisation** (when all array elements are known at the time of creation)

```
int[] numbers = {10, 25, 43, 14, 5};
```

Array indices

- Every position inside an array is identified by its index
- Index numbering starts at 0
 - So the last index of any array is always: length-1

45	65	87	999	651	56	1	2369	444	78
0	1	2	3	4	5	length-1

Value assignment

- To assign/store a value inside the array we have to specify the exact position where it is to be stored:

```
stringArray[0] = "petros";  
numberArray[3] = 12;  
alphabet[1] = 'b';  
dnumbers[2] = 10.2;
```

Accessing stored values

- An array element is accessed by specifying the name of the array and the index (position) of the particular element:

```
//print the 4th element  
System.out.println(integerArray[3]);  
  
//print the 1st element  
System.out.println("Element in 1st position: " + integerArray[0]);  
  
//add the 4th and 6th elements and store result in x  
int x = integerArray[3] + integerArray[5];
```

Going through an entire array

- To do something with all elements of an array we typically use a for loop, with `i` representing the index of the array
- e.g. to prompt the user to fill the array:

```
for (int i = 0; i < numbers.length; i++){  
    System.out.println("Enter value " + i);  
    numbers[i] = scanner.nextInt();  
}
```

To go through the entire array always start at 0 and go up to less than length

- e.g. to print all elements:

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

char arrays → strings

- A character array may be used to initialise a string:

```
char[] name = {'J', 'a', 'v', 'a'};  
  
String s = new String(name);  
  
//s will get the value "Java"
```

Disadvantage

- Biggest disadvantage of arrays is that they **cannot be resized** according to our needs
- We will see in the near future what alternatives we have to address this problem

Additional issues related to Arrays

Arrays and Methods

Passing arrays to methods

- A method can be declared to have an array parameter(s):

```
public static void printArray (int[] array){  
    for (int i = 0; i < array.length; i++){  
        System.out.println(array[i]);  
    }  
}
```

- When the method is called ONLY the array name is passed as an argument (without the brackets):

```
int[] numbers = new int[10];  
...  
printArray(numbers); //calling the printArray method
```

Methods returning arrays

- A method may also be declared to return an array:

```
public static int[] promptForAllGrades(int numberOfStudents){  
    Scanner scanner = new Scanner(System.in);  
  
    int[] allGrades = new int[numberOfStudents];  
  
    for (int i = 0; i < allGrades.length; i++){  
        System.out.println("Enter the grade of student " + i);  
        allGrades[i] = scanner.nextInt();  
    }  
  
    scanner.close();  
    return allGrades;  
}
```

Additional issues related to Arrays

Multidimensional Arrays

Multidimensional Arrays

- A 2-dimensional array is one that contains rows and columns

- Declaring and creating a 2d array:

```
//an array with 2 rows and 4 columns  
int a[][] = new int[3][4];
```

- Declaring and initialising a 2d array:

```
int array[][] = {{10, 20}, {30, 40}};
```

- 10 and 20 are the values of row 0:
array[0][0] and array[0][1]
- 30 and 40 are the values of row 1:
array[1][0] and array[1][1]

A 3x4 2-dimensional array

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Diagram illustrating the indexing of a 3x4 2-dimensional array. The array is represented as a table with rows and columns. The first index is the Row index, and the second index is the Column index. The array name is 'a'.

Going through an entire 2-d array...

- ...you need two nested for loops
 - one to go through all the rows
 - one to go through all the columns
- e.g. to print all elements of a 2-d array:

```
for (int i = 0; i < numbers.length; i++){  
    for (int j = 0; j < numbers[i].length; j++){  
        System.out.println(numbers[i][j]);  
    }  
}
```

This will give you the
length of the row, i.e.
the number of columns

Multidimensional Arrays (cont'd)

- Mind that you can create an array in which every row has a different number of columns (cells)

```
//1st way:  
int b1[][] = new int[2][]; // b1 has 2 rows  
b1[0] = new int[5]; // row 0 of b1 has 5 columns  
b1[1] = new int[3]; // row 1 of b1 has 3 columns  
  
// OR 2nd way:  
int b2[][] = {{ 1, 2, 3, 4, 5 }, { 9, 10, 11 }};
```

(More) Advanced issues

References
and
Reference Parameters

References and Reference Parameters

- Generally, there are two ways to pass arguments to methods
 - Pass-by-value
 - Copy of argument's value is passed to called method
 - Pass-by-reference
 - Caller gives called method direct access to caller's data
 - Called method can directly manipulate this data

References and Reference Parameters - JAVA

- In Java, everything is pass-by-value, but:
 - Unlike a primitive data type variable's name, an object name (in our case an array name) is a reference to the object (i.e. its address)
 - Since object references are passed by value, it means that the data that the reference points to can be changed by a method, but not the reference itself

Example: PassArray.java

Download
PassArray.java and
try it out!

```
public class PassArray {  
    public static void main(String[] args) {  
        int[] array = {1, 2, 3, 4, 5};  
        String output = "Effects of passing entire array by  
reference:\n\nThe values of the original array  
are:\n";  
  
        // append original array elements to String output  
        for (int i = 0; i < array.length; i++)  
            output += array[i] + "\t";  
  
        modifyArray(array); // array passed by reference  
  
        output += "\n\nThe values of the modified array are:\n";  
        for (int i = 0; i < array.length; i++)  
            output += array[i] + "\t";  
    }  
}
```

Declare 5-int array with
initialiser list

Pass array reference by value
to method modifyArray

Example (cont'd): PassArray.java

Pass array[3] (int primitive
data type) by value to
method modifyElement

The original primitive is
left unmodified

```
// attempt to modify array[3]  
output += "\n\narray[3] before modifyElement: " + array[3];  
modifyElement(array[3]);  
output += "\n\narray[3] after modifyElement: " + array[3];  
System.out.println(output);  
} //end of main method  
  
// METHOD: multiply each element of an array by 2  
public static void modifyArray(int[] array) {  
    for (int i = 0; i < array.length; i++)  
        array[i] *= 2;  
}  
  
// METHOD: multiply a number by 2  
public static void modifyElement(int element) {  
    element *= 2;  
}  
} // end class PassArray
```

Method modifyArray
receives array reference

Method modifyElement
receives a primitive's copy

Example: PassArray.java

The OUTPUT

```
C:\Windows\system32\cmd.exe
Effects of passing entire array by reference:
The values of the original array are:
1      2      3      4      5
The values of the modified by the method array are:
2      4      6      8      10
array[3] before modifyElement: 8
array[3] after modifyElement: 8
Press any key to continue . . .
```

The object whose reference is passed to the method is modified

The primitive passed-by-value is unmodified

swap Method

Swapping the position of two values in an array

```
// method that swaps two elements of an array
static void swap(int[] array, int first, int second)
{
    int temp; // temporary holding area for swap

    temp = array[first];
    array[first] = array[second];
    array[second] = temp;
}
```

Check list



- Describe a situation in which you need an array of values
- Can I declare, initialise, and use an array?
- Can I use a for loop to go through an entire array?
- What is biggest disadvantage of arrays?

Check list (cont'd)



- Do I understand what happens when I pass a primitive data type as an argument and what happens when I pass an object (non-primitive data type) as an argument?
- Can a method modify the contents of a passed array?
- Can I declare, initialise, and use multi-dimensional arrays?