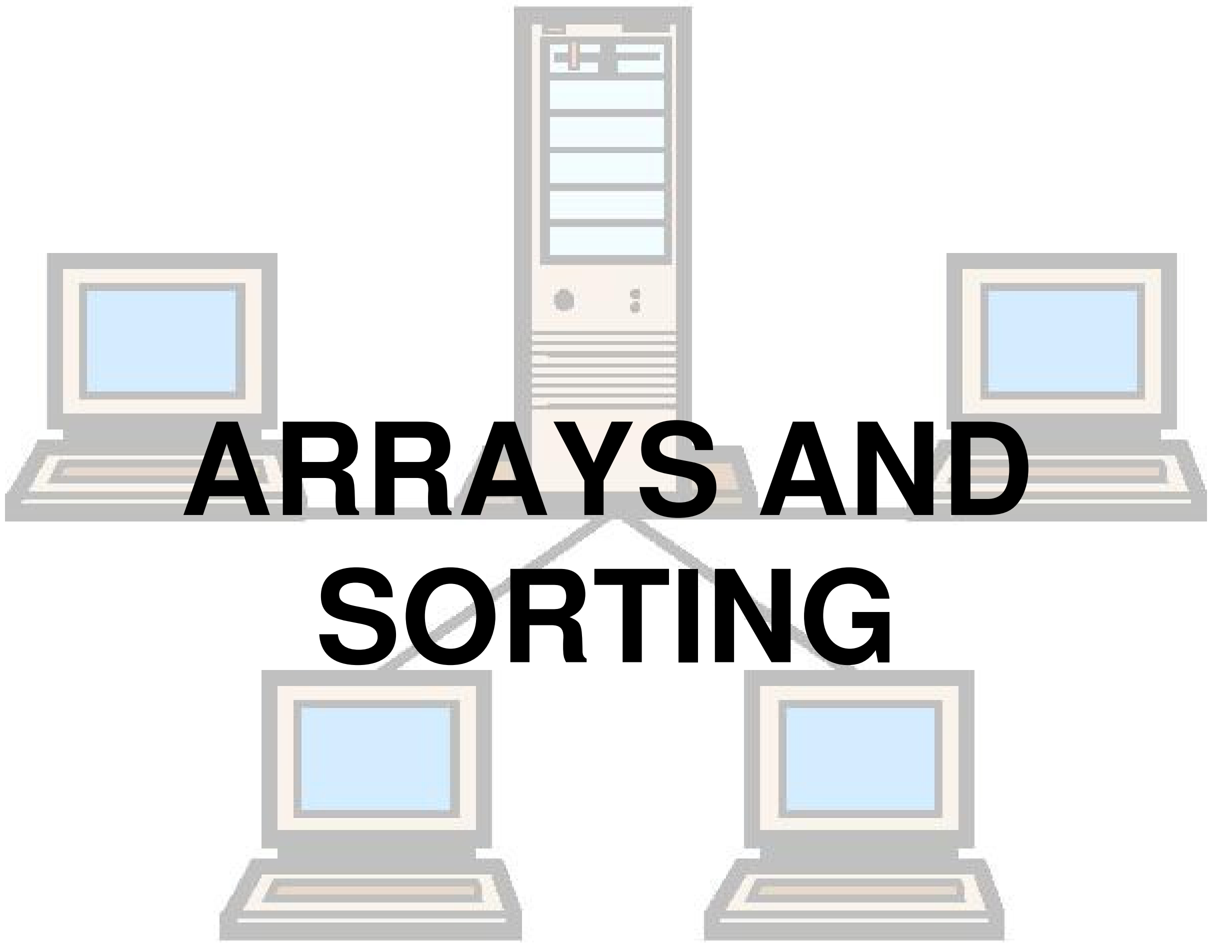


ARRAYS AND SORTING

One-Dimensional Arrays

Introduction

Frequently, we may wish to use the same variable name to refer to a list of related values. In mathematics we use subscripted variables: x_1, x_2, x_3 etc. In Java, we may use similar subscripted or index variables: mark[0], mark[1], mark[2] etc. We call this set of indexed variables a **one-dimensional array**. An array declaration statement is all that is needed to inform the complier that indexed variables are being used.

e.g. `int[] marks = new int [100];`

Memory Location

marks[0]	44
marks[1]	66
marks[2]	20
:	:
marks[99]	96

The above declaration statement indicates that we have a maximum of 100 integers stored under the common name `marks` in 100 consecutive memory locations. (See chart on the right.) The part in square brackets at the end of the line indicates the number of elements in the array.

The reason we use an array (i.e. `marks`) rather than non-subscripted variables (e.g. `mark`) is so that we don't have to declare and process each variable separately but can declare as many similar variables and recall the whole list at any time for later processing. When we use non-subscripted variables, when we enter a new value the previous value will be destroyed so we would not be able to recall all the values entered.

Sample Programs

Enter the following program and run it. Save as **Intro1**.

```
public class Intro1
{
    static BufferedReader stdin = new BufferedReader (new InputStreamReader (System.in));
    static int[] marks = new int [4];

    public static void main (String str[]) throws IOException
    {
        for (int i = 0 ; i < 4 ; i++)
        {
            System.out.print ("Please enter a mark: ");
            marks [i] = Integer.parseInt (stdin.readLine ());
        }

        System.out.println ();
        System.out.println ("Marks");

        for (int i = 0 ; i < 4 ; i++)
        {
            System.out.println (marks[i]);
        }

    }
}
```

Enter the following program and run it. Save as **Intro2**.

```
public class Intro2
{

    public static void main (String str[]) throws IOException
    {
        BufferedReader stdin = new BufferedReader (new InputStreamReader (System.in));

        final int MAX = 3;
        int[] X = new int [MAX];
        int[] Y = new int [MAX];

        for (int i = 0 ; i < MAX ; i++)
        {
            System.out.print ("Please enter an integer: ");
            X [i] = Integer.parseInt (stdin.readLine ());
        }

        System.out.println ("\tX\tY");

        for (int i = 0 ; i < MAX ; i++)
        {
            Y [i] = 3 * X [i];
            System.out.println ("\t" + X [i] + "\t" + Y [i]);
        }

    }
}
```

Notice how the upper bound is defined. The use of the `final` statement increases the flexibility of the program, particularly if you had to use a number of statements involving MAX.

Array Exercises

Level 1

Create a menu-driven main method that will run any void method described below.
Save as **Array1.java**.

1. Declare a global variable which is an array of 10 integers.

2. Write a void method called ***InitializeArray*** which will assign the value -1 to every element of the array.

3. Add a void method called ***EnterFromKeyboard*** which will enter UP TO 10 integers and store them in the array.

4. Change the main method so it will call the first two void methods (*InitializeArray* and *EnterFromKeyboard*) then display the menu.

5. Add a void method called ***CountWh*****ole** which will calculate and display the number of whole numbers entered into the array.

6. Add a void method called ***Display*** which will display the list of inputted integers in the order entered.
   e.g. `The integers in order entered  are 8  12  32  43  14  12.`

7. Add a void method called ***DisplayReverse*** which will display the list of inputted integers in reverse order entered.
   e.g. `The integers in reverse order is 12  14  43  32  12  8.`

8. Add a void method called ***Sum*** that will calculate and display the sum of all the numbers entered.

9. Add a void method called ***Average*** that will calculate and display the average of all the numbers entered correct to 1 decimal place.

10. Add 2 void methods called ***FindMax*** and ***FindMin*** which will calculate and display the maximum number and the minimum number stored in the array respectively.

11. Add a void method called ***Search*** that will display the position(s) in the array a specific number occupies.
    e.g. `The number 12 is found in the following positions: 2, 5.`

12. Add a loop so that the user can make other choices from the menu for the numbers that the user entered

Level 2

Part A: Modify **Array1.java** by changing the following. Save as ***Array2a.java.***

1. Change the void methods ***FindMax*** and ***FindMin*** so that the number of occurrences will also be stated.

```
e.g.    The maximum integer is 126.     # of occurrences is 1.
        The minimum number  is 2.       # of occurrences is 3.
```

2. Change the void method ***Search*** so that it will determine if a specific number is NOT found in the array.

Part B: This is a new file, call it ***Array2b.java.*** Create a menu driven main method that will run any void method described below. Use only local variables.

1. Write a void method called ***CountOccurrences*** which will prompt the user to enter a natural number from 1 to 10 (stop entering numbers when the user enters a number outside the range). The void method will output the number of the times each number was entered.

```
e.g.    Enter a number: 5
        Enter a number: 2
        Enter a number: 3
        Enter a number: 2
        Enter a number: 2
        Enter a number: -5

        Number      # of occurrences
          1              0
          2              3
          3              1
          :              :
         10              0
```

2. Add void method ***CountOccurrences*****2** which is identical to ***CountOccurrences*** except that it counts numbers between 15 and 25 inclusive.

3. Add a void method called ***Totals*** which will prompt the user to enter whole numbers from 0 to 99 (stop entering numbers when the entered number is outside the range). The void method will output the total (sum) of all numbers entered less than 10, the total of all numbers entered in the teens (10 to 19), the total of all numbers entered in the 20s, ... and the total of all numbers entered in the 90s.

4. Add a void method called ***Totals*****2** which will look similar to ***Totals***, but calculates the totals (sums) slightly differently. The void method will output the total of <u>all</u> numbers entered less than 10, the total of <u>all</u> numbers entered less than 20, the total of <u>all</u> numbers entered less than 30, ... and the total of <u>all</u> numbers entered less than 100 (so 88 will be included in both totals for numbers less than 90 and less than 100).

Level 3

1. Change **Array2a.java** so that you have no global variables, i.e. the arrays are passed to each method as parameters. Save as **Array2d**.

The following is a new file; call it **Array3a.java**. Create a menu-driven main method that will run any method described below.

2. Create a method called ***NoDuplicates*** which will prompt the user to enter 7 unique integers. As the user enters the numbers, ask them to re-enter a number if it has been entered previously. Output the 7 unique numbers.

3. Create a method called ***RandomNoDuplicates*** which will output 10 unique integers all of which fall within a user-specified range.

4. Your grade 10 counterparts are programming in Turing, where colours are specified by integers (e,g, red = 12). If we ask the user to enter a colour, rather than making them enter a number, we could allow them to enter the name of a colour, stored as a string. This string could be changed to the appropriate integer using an array as a look-up table. In order to do this, all the names of colours (as strings) could be stored in the position corresponding with the integer corresponding with the integer colour value. You would then just have to look through the array to find the string and determine what position it is stored in. Create a method called ***DetermineColour*** which will return the Turing colour integer value of a colour given as a string.

5. Create a method called ***EnterAndCount*** that will ask the user to enter 4 numbers (allowing no duplicates) from 1 and 100 that they want to look for and count. The program will then open a text file (the user should enter the filename) and count the number of times those numbers appear in the text file. Output each number and the number of asterisks (*) equal to the number of times that number was found in the text file. Output the number which had the most asterisks associated with it.

e.g.
```
 9: ******
22: **
57: *
77: ***
The most common number chosen was 9.
```

6. Five divers are to compete in a diving competition. Create a method called ***RandomOrder*** which will output the names of these 5 divers in a random diving order.

7. Create a method called ***ThreeRandomOrder*** which will output 3 random orders with different divers going first and last each time. There are 5 divers.

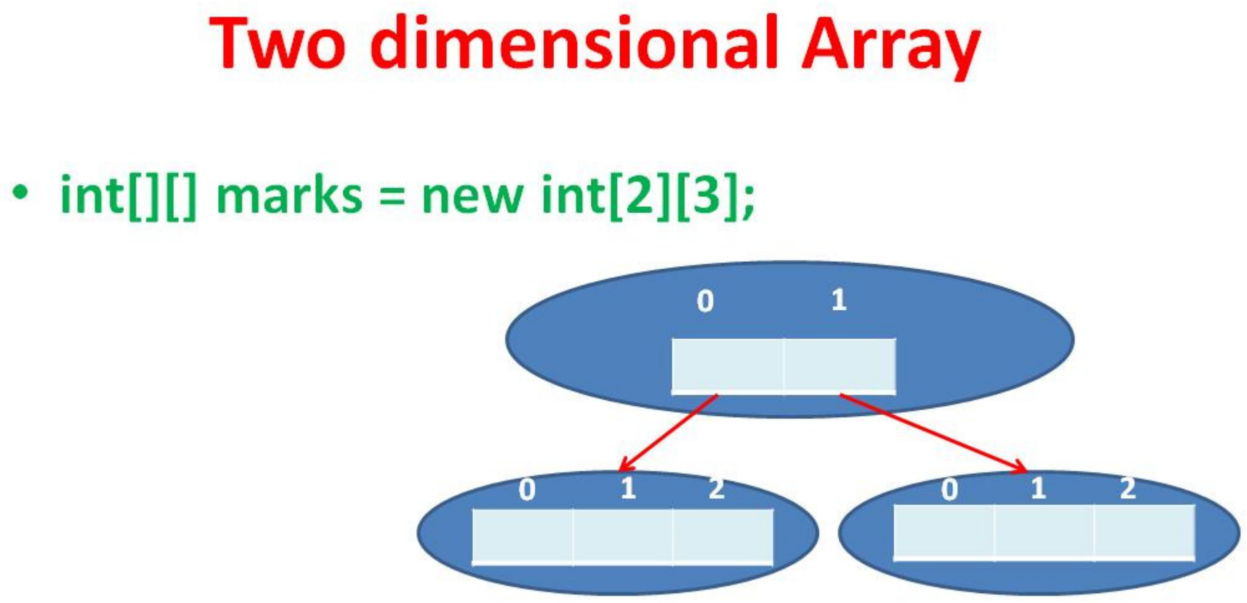
Two-Dimensional Arrays

Whereas in other programming languages **Two-Dimensional Arrays** are often viewed as tables or chart, with rows and columns, in Java they are simply arrays of arrays. Here is the syntax for two-dimensional arrays:

```
int[][] array = new int[MAX_X][MAX_Y];
```

The first bracket denotes the number of elments in the first array, while the second bracket denotes the number of arrays within each array (the second dimension).

The example below demonstrates the exact meaning of “arrays of arrays”:



Two dimensional Array

- int[][] marks = new int[2][3];

Level 4

The following are new files. **DO NOT USE GLOBAL VARIABLES.**

1. Write the following methods for a 3-by-4 array:

   a) the method should fill the array, one “row” at a time; input can be from the keyboard or from a text file (the user should enter the filename) and the programmer decides which dimension is “rows” and which is “columns”

   b) the method should display the array on the screen as it was entered (“rows” across and “columns” down)

   c) the method should display the array where the columns appear across the screen and the rows down the screen; this is called the transpose of an array

   e.g.
   ```
   Original Order           Transposed Order
   3   4   5   3            3   1   5
   1   2   3   8            4   2   6
   5   6   7   9            5   3   7
                            3   8   9
   ```

   d) the method should display the sum of each column at the end of each column and the sum of each row at the end of each row

   Save as **Array41.java**.

2. Write a program that will determine class marks based on tests. The marks must be stored in a two-dimensional array. The data should be read from a text file where the name of the student is followed by 4 test marks. Your methods should calculate and output in table form the students’ names, their marks, the average on each test and each student’s average. Save as **Array42.java**.

3. Create your own NEW application which requires arrays as part of the solution. Save as **Array43.java**.

Sorting Exercises

1. Given the file “Unsort.txt” containing a list of names and write a program that would open the file and do the following:

   a) reads the information into a suitable array and outputs the list to screen
   b) sorts the array of names records alphabetically (ascending order) and displays on the screen
   c) write the sorted list to another file called “sorted1.txt”
   (Hint: since you are working with 2 different files, use two different file buffer objects)

   Save as **Sort1.java**.

2. Given the file “login.txt” containing a list of user names and the number of times each user has logged in, do the following:

   a) read the information and output the list to screen
   b) sort the login names according to the person who logged in the most to the person who logged in the least and display the sorted list on the screen
   c) write the sorted list to another file called “sorted2.txt”

   Save as **Sort2.java**.

3. Given the file “Unsort3.txt” containing a list of names (the first name of a person followed by their last name), write a program that will open the file and complete the following:

   a) read the first names into one array and the last names into **another** array
   b) sort the first names in alphabetical order and display on the screen the sorted list (first name and corresponding last name), making sure the last name matches the first name
   c) notice that the list is not sorted correctly because some of the first names are the same and correct the problem
   d) write the sorted list (first name and last name) to another file called “sort3.txt”

   Save as **Sort3.java**.