CHAPTER 8

ARRAYS

Suppose a class contains 50 students, and we want to store their roll numbers, we need 50 separate variables for storing the roll numbers, as shown here:

```
int rno;
int rno1;
int rno2;
:
int rno49;
```

Now to store roll numbers into these variables, we need another 50 statements. Imagine writing 100 statements just to store the roll numbers of students. On the other hand, if we have a single variable which can represent all these 50 variables, it would be very useful to us. Such a variable is called an 'array'.

An array represents a group of elements of same data type. It can store a group of elements. So, we can store a group of int values or a group of float values or a group of strings in the array. But we can not store some int values and some float values in the array.

The advantage of using arrays is that they simplify programming by replacing a lot of statements by just one or two statements. In C /C++, by default, arrays are created on static memory unless pointers are used to create them. In Java, arrays are created on dynamic memory, i.e., allotted at runtime by JVM.

Important Interview Question

On which memory, arrays are created in Java?

Arrays are created on dynamic memory by JVM. There is no question of static memory in Java; every thing (variable, array, object etc.) is created on dynamic memory only.

Types of Arrays

Arrays are generally categorized into two parts as described here:

- ❑ Single dimensional arrays (or 1D arrays)
- ❑ Multi dimensional arrays (or 2D, 3D, ... arrays)

Single Dimensional Arrays (1D array)

A one dimensional (1D) or single dimensional array represents a row or a column of elements. For example, the marks obtained by a student in 5 different subjects can be represented by a 1D array, because these marks can be written as a row or as a column.

Creating a Single Dimensional Array

There are some ways of creating a single dimensional array as mentioned here:

- ❑ We can declare a one dimensional array and directly store elements at the time of its declaration, as:

```
int marks[ ] = {50, 60, 55, 67, 70};//declare marks[] and initialize with  5
                                    //elements
```

Here, 'int' represents integer type elements which can be stored into the array, and the array name is 'marks'. To represent a one dimensional array, we should use a pair of square braces [ ] after the array name. Then the actual elements (integers) are mentioned inside the curly { and } braces. Now JVM creates 5 blocks of memory as there are 5 elements being stored into the array. These blocks of memory can be individually referred to as marks[0], marks[1], marks[2],...marks[4]. Here, 0,1,2,...4 is called 'index' of the array. It refers to the element position in the array. A one dimensional array will have only one index. In general, any element of the array can be shown by writing marks[i], where i = 0,1,2,...4. Please see the Figure 8.1.

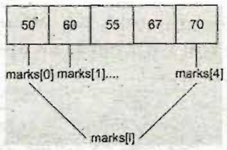

Figure 8.1 Arrangement of elements in a 1D array

- ❑ Another way of creating a one dimensional array is by declaring the array first and then allotting memory for it by using new operator.

```
int marks[ ];   //declare marks array
marks = new int[5];  //allot memory for storing 5 elements
```

These two statements can also be written by combining them into a single statement, as:

```
int marks[ ] = new int[5];
```

Here, we should understand that JVM allots memory for storing 5 integer elements into the array. But there are no actual elements stored in the array so far. To store the elements into the array, we can use statements like these in the program:

```
marks[0]= 50;
marks[1]= 60;
marks[2]= 55;
marks[3]= 67;
marks[4]= 70;
```

Or, we can pass the values from keyboard to the array by using a loop, as given here:
for(int i=0; i<5; i++)

```
{
      //read integer value from the keyboard and store into marks[i]
      marks[i] = Integer.parseInt(br.readLine());
}
```

Let us examine some more examples for 1D array:

- `float salary[ ] = {5670.55f, 12000f, 4500.75f, 3000.50f, 9050f};`
- `float salary[ ] = new float[50];`
- `char ch[ ] = {'a','b','c','d','e','f'};`
- `char ch[ ] = new char[6];`
- `String names[ ]= {"Raju", "Vijay", "Gopal", "Kiran"};`
- `String names[ ]= new String[10];`

Alternative Way of Writing One Dimensional Array

While writing a one dimensional array, we can write the pair of square braces before or after the array name, as:

- `float salary[ ] = {5670.55f, 12000f, 4500.75f, 3000.50f, 9050f};`
- `float[ ] salary = {5670.55f, 12000f, 4500.75f, 3000.50f, 9050f};`

The preceding statements are same and valid. Similarly, the following statements are also same:

- `String names[ ]= new String[10];`
- `String[ ] names = new String[10];`

Program 1: Write a program to create a 1D array and read its elements by using a loop and display them one by one.

To read and display a 1D array:

```
class Arr1
{
        public static void main(String args[])
        {
                //declare and initialize the array
                int arr[] = {50, 60, 55, 67, 70};
                //display all the 5 elements
                for(int i=0; i<5; i++)
                {
                        System.out.println(arr[i]);
                }
        }
}
```

Output:

```
C:\> javac Arr1.java
C:\> java Arr1
50
60
55
67
70
```

Usually a single loop is suitable to read the elements from a 1D array or to store elements into a 1D array. In the following program, let us take the marks obtained by a student from the keyboard and find their total and percentage of marks. Think about the logic to find the total marks. For this purpose, we take a variable 'tot' and initialize it to 0, as initially the total marks will be 0, like this:

```
int tot =0;
```

Then, add the elements one by one to 'tot' and store the result in 'tot', like this:

```
tot = tot + marks[i];
This can be also written as:
tot += marks[i];
```

Now, let us think about the logic to find the percentage. If the maximum mark in any subject is 100, then percentage can be obtained by dividing the total marks (tot) by the number of subjects (n), thus:

```
        float percent = tot/n;
```

Since, tot and n are both integers, when an integer value is divided by another integer, by default only int value will be returned. This means if tot = 282 and n = 5, then percent value will be only 56.0. This is definitely wrong result. So to get the correct result, we should convert at least one of the variables tot or n into float type. This can be achieved by using casting, as given here:

```
        float percent = (float) tot/n;
```

Here, we are converting the type of tot into float. So float divided by int value gives us correct float value. Now, percent value will be 56.4. Note that the data type of tot, i.e., 'int' is converted into 'float' type here. Converting a data type into another data type is called 'type casting' or simply 'casting'. And for this purpose, we wrote (float) before tot variable. Writing data type like this, in between simple braces before a variable or a method is called 'cast operator'.

Let us have a glance on another program.

Program 2: Write a program which accepts the marks of a student into a 1D array from the keyboard and finds total marks and percentage.

```
//Total marks and percentage
import java.io.*;
class Arr2
{
        public static void main(String args[]) throws IOException
        {
                //to accept data from keyboard
                BufferedReader br = new BufferedReader(new
                InputStreamReader(System.in));

                //ask how many subjects
                System.out.print("How many subjects? ");
                int n = Integer.parseInt(br.readLine());

                //create 1D array with size n
                int[] marks = new int[n];

                //store elements into the array
                for(int i=0; i<n; i++)
                {
                        System.out.print("Enter marks: ");
                        marks[i]= Integer.parseInt(br.readLine());
                }

                //find total marks
                int tot = 0;
                for(int i=0; i<n; i++)
                tot += marks[i];

                //display total marks
                System.out.println("Total marks= "+ tot);

                //find percentage
                float percent = (float)tot/n;
                System.out.println("Percentage= "+ percent);
        }
}
```

Output:

```
C:\> javac Arr2.java
```

```
C:\> java Arr2
How many subjects? 5
Enter marks: 50
Enter marks: 60
Enter marks: 55
Enter marks: 50
Enter marks: 67
Total marks= 282
Percentage=  56.4
```

Let us write a program to accept a group of elements into an array and sort them into ascending order. For this purpose, let us use 'bubble sort' technique. In this technique, all the n elements from 0 to n-1 are taken and the first element of the array: arr[j] is compared with the immediate element arr[j+1]. If arr[j] is bigger than arr[j+1], then they are swapped (interchanged) since in ascending order, we expect the smaller elements to be in the first place. When two elements are interchanged, the number of elements to be sorted becomes lesser by 1. When there are no more swaps found, then the 'flag' will become false and we can abort sorting. This logic is implemented in the following Program 3.

Program 3: Write a program which performs sorting of group of integer values using bubble sort technique.

Sort a group of integers into ascending order:

```
import java.io.*;
class Sort
{
        public static void main(String args[]) throws IOException
        {
                //to accept data from keyboard
                BufferedReader br = new BufferedReader(new
                InputStreamReader(System.in));

                //create an int type array
                System.out.print("How many elements? ");
                int n = Integer.parseInt(br.readLine());
                int arr[] = new int[n];

                //accept integer elements into the array
                for(int i=0; i<n; i++)
                {
                        System.out.print("Enter int: ");
                        arr[i] = Integer.parseInt(br.readLine());
                }

                //use bubble sort technique to sort the integers
                int limit = n-1; //elements from 0 to n-1
                boolean flag = false; //if it is true, swapping done
                int temp; //temporary variable

                for(int i=0; i<limit; i++)
                {
                        for(int j=0; j<limit-i; j++)
                        {
                                //if first element is bigger than second one, then
                                swap
                                if(arr[j] > arr[j+1])
                                {
                                        temp = arr[j];
                                        arr[j] = arr[j+1];
                                        arr[j+1] = temp;
                                        flag = true; //true -> swapping done
                                }
                        }

                        if(flag==false) break; //no swapping, so come out
                        else flag = false; //assign initial value
```

```
                    }
                    //display the sorted array
                    System.out.println("The sorted array is: ");
                    for(int i=0; i<n; i++)
                    System.out.println(arr[i]);
            }
    }
```

Output:

```
    C:\> javac Sort.java
    C:\> java Sort
    How many elements? 5
    Enter int: 50
    Enter int: 23
    Enter int: 11
    Enter int: 99
    Enter int: 23
    The sorted array is:
    11
    23
    23
    50
    99
```

Multi Dimensional Arrays (2D, 3D,... arrays)

Multi Dimensional arrays represent 2D, 3D,...arrays which are combinations of several earlier types of arrays. For example, a two dimensional array is a combination of two or more (1D) one dimensional arrays. Similarly, a three dimensional array is a combination of two or more (2D) two dimensional arrays. Let us understand the two dimensional arrays now.

Two dimensional arrays (2D array)

A two dimensional array represents several rows and columns of data. For example, the marks obtained by a group of students in five different subjects can be represented by a 2D array. If we write the marks of three students as:

- 50, 60, 55, 67, 70
- 62, 65, 70, 70, 81
- 72, 66, 77, 80, 69

The preceding marks represent a 2D array since it got 3 rows (no. of students) and in each row 5 columns (no. of subjects). There are totally 3X5 = 15 elements. We can take the first row itself as a 1D array. Similarly the second row is a 1D array and the third row is another 1D array. So the preceding 2D array contains three 1D arrays within it.

Creating a (2D) Two Dimensional Array

There are some ways of creating (2D) two dimensional array as mentioned here:

- We can declare a two dimensional array and directly store elements at the time of its declaration, as:

```
    int marks[ ][ ] = {{50, 60, 55, 67, 70},
                       {62, 65, 70, 70, 81},
                       {72, 66, 77, 80, 69}};
```

Here, 'int' represents integer type elements stored into the array, and the array name is 'marks'. To represent a two dimensional array, we should use two pairs of square braces [ ][ ] after the array name. Each row of elements should be written inside the curly braces { and }. The rows and the elements in each row should be separated by commas. There are three rows and five columns in each row, so the JVM creates 3X5= 15 blocks of memory as there are 15 elements being stored into the array. These blocks of memory can be individually referred to as mentioned here:

```
marks[0][0], marks[0][1], marks[0][2], marks[0],[3], marks[0][4]
marks[1][0], marks[1][1], marks[1][2], marks[1],[3], marks[1][4]
marks[2][0], marks[2][1], marks[2][2], marks[2],[3], marks[2][4]
```

By observing the preceding elements, we can understand the rows which are starting from 0 to 2 and the columns are starting from 0 to 4. So any element can be referred in general as marks[i][j], where i represents row position and j represents column position. Thus, a two dimensional array has two indexes: i and j. Similarly we can expect three indexes in case of a three dimensional array. See the Figure 8.2.

	j=0	j=1	j=2	j=3	j=4
i = 0	50	60	55	67	70
i = 1	62	65	70	70	81
i = 2	72	66	77	80	69

Figure 8.2 Arrangement of elements in a 2D array

- Another way of creating a two dimensional array is by declaring the array first and then allotting memory for it by using new operator.

```
int marks[ ][ ];    //declare marks array
    marks = new int[3][5];  //allot memory for storing 15 elements
```

The preceding two statements can be written by combining them into a single statement, as:

```
int marks[ ][ ] = new int[3][5];
```

Here, JVM allots memory for storing 15 integer elements into the array. But there are no actual elements stored in the array so far. We can store these elements by accepting them from the keyboard or from within the program also.

Let us take some more examples for 2D arrays:

- float x[ ][ ] = {{1.1f, 1.2f, 1.3f, 1.4f},
  {2.1f, 2.2f, 2.3f, 2.4f},
  {3.1f, 3.2f, 3.3f, 3.4f}};
- double d[ ][ ] = {{20.2, -5.5}, {15.5, 30.331}};
- byte b[ ][ ] = new byte[20][50];
- String str[ ][ ] = new String[10][20];

Alternative Way of Writing Two Dimensional Arrays

While writing a two dimensional array, we can write the two pairs of square braces before or after the array name, as:

```
String str[ ][ ] = new String[10][20];
String[ ][ ] str = new String[10][20];
```

Let us write a program to take a 2D array and display its elements in a matrix form. A matrix represents a group of elements arranged in several rows and columns. So we can take a 2D array as a matrix.

Program 4: Write a program to take a 2D array and display its elements in the form of a matrix. To display the elements of 2D array, we use two for loops, the outer for loop represents the rows and the inner one represents the columns in each loop.

```
//Displaying a 2D array as a matrix
class Matrix
{
	public static void main(String args[])
	{
	//take a 2D array
	float x[ ][ ] = {{1.1f, 1.2f, 1.3f, 1.4f},
		{2.1f, 2.2f, 2.3f, 2.4f},
		{3.1f, 3.2f, 3.3f, 3.4f}};
		//read and display the array elements
		System.out.println("In matrix form");
		for(int i=0; i<3; i++)    //rows
		{
			for(int j=0; j<4; j++) //columns in each row
			{
				System.out.print(x[i][j]+"\t");
			}
			System.out.println();  //next line
		}
	}
}
```

Output:

```
C:\> javac Matrix.java
C:\> java Matrix
 1.1 1.2     1.3     1.4
 2.1 2.2     2.3     2.4
 3.1 3.2     3.3     3.4
```

A two dimensional array can be handled by using two loops. Similarly, a three dimensional array can be handled using three loops.

Let us write a program to find the transpose of a given matrix. The transpose of a matrix is defined as the matrix obtained by converting the rows as columns and columns as rows. Suppose we have a matrix with 3 rows and 4 columns. Its transpose matrix will have 4 rows and 3 columns.

In this program, we are using Scanner class to accept the input from the keyboard. First, connec keyboard to Scanner, as:

```
Scanner sc = new Scanner(System.in);
```

Now, whatever the int values supplied from the keyboard can be accepted by using sc.nextInt(method. This method accepts next integer value from the Scanner. So, by using this method in a loop, it is possible to accept all the integer elements of the array from the keyboard, as shown here:

```
for(int i=0; i<r; i++)
	for(int j=0; j<c; j++)
		arr[i][j] = sc.nextInt();
```

The outer loop is for rows and the inner loop represents columns. In these loops, one by one th elements are received by sc.nextInt() method and are assigned to the array `arr[i][j]`.

Program 5: Write a program which accepts elements of a matrix and displaying its transpose.

```
//Transpose of a matrix.
import java.io.*;
import java.util.Scanner;
class Transpose
{
	public static void main(String args[]) throws IOException
	{
		//use Scanner to accept data from keyboard
		Scanner sc = new Scanner(System.in);
```

```
                //accept rows, columns of a matrix
                System.out.print("Enter rows, columns? ");
                int r = sc.nextInt();
                int c = sc.nextInt();

                //create an array with size [r][c]
                int arr[ ][ ] = new int[r][c];

                //accept a matrix from keyboard
                System.out.println("Enter elements of matrix: ");

                for(int i=0; i<r; i++)
                for(int j=0; j<c; j++)
                arr[i][j] = sc.nextInt();

                System.out.println("The transpose matrix: ");

                //take columns as rows and vice versa and display
                for(int i=0; i<c; i++)
                {
                        for(int j=0; j<r; j++)
                        {
                                System.out.print(arr[j][i]+ "  ");
                        }
                        System.out.print("\n");
                }
        }
}
```

Output:

```
C:\> javac Transpose.java
C:\> java Transpose
Enter rows, columns? 3 4
Enter elements of matrix:
1  2  3  4
5  6  7  8
9  9  -1  2
The transpose matrix:
1  5  9
2  6  9
3  7  -1
4  8  2
```

Three dimensional arrays (3D array)

We can consider a three dimensional array as a combination of several two dimensional arrays. This concept is useful when we want to handle group of elements belonging to another group. For example, a college has 3 departments: Electronics, Computers and Civil. We want to represent the marks obtained by the students of each department in 3 different subjects. We can write these marks as shown here:

Electronics department:

```
Student1 marks: 50,51,52
Student2 marks: 60,61,62
```

Computers department:

```
Student1 marks: 70,71,72
Student2 marks: 80,81,82
```

Civil department:

```
Student1 marks: 65,66,67
Student2 marks: 75,76,77
```

To store all these marks, department-wise, we need a three dimensional array as shown here:

```
int arr[ ][ ][ ] = {{{50,51,52}, {60,61,62}},
                    {{70,71,72}, {80,81,82}},
                    {{65,66,67}, {75,76,77}}};
```

Here, three pairs of square braces represent a 3D array and there are 3 rows representing the 3 departments. In each department, there are again two groups representing the marks of two students. Observe how the { and } curly braces are used to represent each group. Other ways of creating three dimensional arrays:

```
char x[ ][ ][ ] = new char[10][5][20];
float [ ][ ][ ]f = new float[5][6][10];
```

Program 6: Write a program in which we take a 3D array which consists of department wise student marks. There are 3 departments and in each department, there are 2 students and each student has marks in 3 subjects. We want to calculate total marks of each student.

```
//Three dimensional array
class ThreeD
{
        public static void main(String args[])
        {
                //declare three vars
                int dept, student, marks, tot=0;

                //take the marks of students in a 3D array
                int arr[ ][ ][ ] = {{{50,51,52}, {60,61,62}},
                                    {{70,71,72}, {80,81,82}},
                                    {{65,66,67}, {75,76,77}}};

                //display the marks from 3D array
                for(dept=0; dept<3; dept++)
                {
                        System.out.println("Department "+(dept+1)+": ");
                        for(student=0; student<2; student++)
                        {
                                System.out.print("Student "+(student+1)+" marks: ")
                                for(marks=0; marks<3; marks++)
                                {
                                System.out.print(arr[dept][student][marks]+" ");
                                tot += arr[dept][student][marks];
                                }
                                System.out.println("Total: "+ tot); //display total
                                marks of a student
                                tot =0;  //reset total to 0
                        }
                        System.out.println();
                } //end of for loops

        }
}
```

Output:

```
C:\> javac ThreeD.java
C:\> java ThreeD
Department 1:
Student 1 marks: 50  51  52  Total: 153
```

```
        Student 2 marks: 60  61  62  Total: 183

        Department 2:
        Student 1 marks: 70  71  72  Total: 213
        Student 2 marks: 80  81  82  Total: 243

        Department 3:
        Student 1 marks: 65  66  67  Total: 198
        Student 2 marks: 75  76  77  Total: 228
```

arrayname.length

If we want to know the size of any array, we can use the property 'length' of an array. arrayname.length returns an integer number which represents the size of an array.

For example, take the array arr[ ]with size 10 and there are three elements in it, as:

```
        int arr[ ] = new int[10];  //size is 10
        arr[0]= 50, arr[1]=55, arr[2]=60;  //number of elements is 3.
```

Now, arr.length gives 10. The array arr[ ] contained 3 elements, but 'length' property does not give the number of elements of the array. It gives only its size.

But, in case of a two or three dimensional array, 'length' property gives the number of rows of the array, as given here:

```
        int arr[ ][ ] = new int[5][10];
        int arr[ ][ ][ ] = new int[5][7][12];
```

In preceding two cases, arr.length gives 5.

Command Line Arguments

We have already learned in the earlier chapters that command line arguments represent the values passed to `main()` method. To catch and store these values, `main()` has a parameter, String `args[ ]` as:

```
        public static void main(String args[ ])
```

Here, `args[ ]` is a one dimensional array of String type. So it can store a group of strings, passed to `main()` from outside by the user. The user should pass the values from outside, at the time of running the program at command prompt, as:

```
        C:\> java  Prog  11  22  Vikas
```

The three values passed to `main()` at the time of running the program are 11, 22 and Vikas. These three values are automatically stored in the `main()` method's `args[ ]` in the form of strings. This is because, `args[ ]` is a String type array. Thus 11 is stored as a string in `args[0]`, 22 is stored as a string in args[1] and Vikas is stored in `args[2]`. See the Figure 8.3.

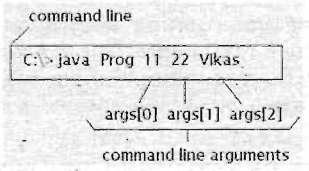

Figure 8.3 Command line and Command line arguments

```
String names[ ] = {"Gopi", "Kamal", "Vinay", "Neeta Jain"};
```

And this array is sent to main() method of Class1 as:

```
Class1.main(names);
```

Thus, if we pass the name of the array, the entire array is passed to `main()` method.

Program 9: Write a program which calls the `main()` method of Class1 from Class2. In this, we pass `names[ ]` array to `main()` method at the time of call. This `names[ ]` is copied into args[ ] and hence we can display the contents of `names[ ]` in the `main()` method of Class1. Since we are calling Class1 from Class2, first of all Class2 should be executed by JVM, so we give this program name as Class2.java.

```
//calling main() of a Class1 from another class: Class2
class Class1
{
	public static void main(String args[ ])
	{
		//args contains names, display them
		for(String s: args)   //we are using for-each loop here
		System.out.println(s);
	}
}
class Class2
{
	public static void main(String args[ ])
	{
		//take a string type array
		String names[] = {"Gopi", "Kamal", "Vinay", "Neeta Jain"};

		//call main() of Class1  and pass the names array
		Class1.main(names);
	}
}
```

Output:

```
C:\> javac Class2.java
C:\> java Class2
Gopi
Kamal
Vinay
Neeta Jain
```

Conclusion

In this chapter, you learnt that arrays are very useful for a programmer to handle a group elements easily. But the restriction is that all the elements which are to be stored into the ar should be of same data type. To store, retrieve and process elements of a 1D array, a single looj much suitable. Similarly to work with a 2D array, two loops: a loop inside another loop convenient. The main() method that is written in all Java programs has a parameter: args[ ], wh is a String type array. The purpose of this array is to accept input from the user through comma line. The same input can be used in the program by the programmer. We also discussed that e array has a property, arrayname.length which gives the size of the array.