**Introduction to Array in Java**

In this digital world, storing information is a very crucial task to do as a programmer. Every little piece of information needs to be saved in the memory location for future use. Instead of storing hundreds of values in different hundreds of variables, using an array, we can **store all these values in just a single variable**. To use that data frequently, we must assign a name, which is done by variable.

An array is a **homogenous non-primitive** data type used to save multiple elements (having the same data type) in a particular variable.

Arrays in Java can hold primitive data types (Integer, Character, Float, etc.) and non-primitive data types(Object). The values of primitive data types are stored in a memory location, whereas in the case of objects, it's stored in heap memory.

Assume you have to write a code that takes the marks of five students. Instead of creating five different variables, you can just create an array of lengths of five. As a result, it is so easy to **store and access data from only one place**, and also, it is **memory efficient**.

**How do Arrays in Java work?**

As discussed above, an array is a data structure or container that stores the same typed elements in a sequential format.

Let's use the above example to know how arrays work in java. This is how we can implement a program to store five subject marks of a student:

int mark1 = 78;

int mark2 = 84;

int mark3 = 63;

int mark4 = 91;

int mark5 = 75;

But what if we store all these five variables in one single variable?

marks[0] = 78;

marks[1] = 84;

marks[2] = 63;

marks[3] = 91;

marks[4] = 75;

All above mentioned five variables could be accessed by the index given in the square bracket, which is known as index-based variables.

The **index always starts from 0** by default. So, if we have an array of 5 elements, it will have an index range from 0 to 4 by default.

In the above example, the first element is stored at index 0 and the last one at the 4th index. The below-mentioned diagram shows how the array got placed in computer memory.

**How to Create and Initialize Array in Java?**

In every program, to use an array, we need to create it before it is used. We require two things to create an array in Java - data type and length.

As we know, an array is a homogenous container that can only store the same type of data. Hence, it's required to specify the data type while creating it. Moreover, we also need to put the number of elements the array will store.

**Syntax:**

Unlike other programming languages, while declaring an array in java, we can put [] both before and after the variable name.

The **new** keyword is crucial to creating an array. Without using it, we cannot make array reference variables.

datatype[] arrayName = new datatype[length]; OR

datatype arrayName[] = new datatype[length];

*//[length] = Length of the array*

**Example:**

*// Both will create an int array with a length of 5.*

int[] myArray = new int[5];

int myArray1[] = new int[5];

Explore free courses by our top instructors[View All](https://www.scaler.com/topics/courses/)

TARUN LUTHRA

Java Course - Mastering the Fundamentals

52k+ enrolled

RAHUL JANGHU

Python Certification Course: Master the essentials

37k+ enrolled

PRATEEK NARANG

C++ Course: Learn the Essentials

27k+ enrolled

35,262+ learners have attended these Courses.

**We can Initialize Arrays in Java in two Different ways -**

1. **Initialize using index -** Once an array is created successfully, we can assign the value to any specific element using its index. Un-assigned elements automatically take the default value of the array's data type. In this case, 0 for the integer data type. On the other hand, if we try to enter more values than the size of the array, it will raise an ArrayIndexOutOfBoundsException.

**Example:**

int[] myArray = new int[5]; *// array declared*

myArray[0] = 10; *// assigned 10 at index 0*

myArray[2] = 37; *// assigned 37 at index 0*

Array will be -

10 0 37 0 0

1. **Initialize while declaring -** We can also initialize the whole array while declaring it just by using curly brackets {}. The string must be in quotes, whereas int, float, etc., do not require that.

**Example:**

int[] myArray = {1, 2, 3, 4, 5}; *// For integer type*

String[] myArray1 = {"A", "B", "C", "D"}; *// For string type*

**How to Change an Array Element?**

To change a value at a specific position in Java arrays, refer to an index. By providing index value, we tell the Java compiler where we want to change the value.

For this, we need to use a Square bracket [] with an index number inside it. If somehow we entered the index number bigger than the array length, then an exception occurs of type ArrayIndexOutOfBoundsException (We will learn more about it in this article).

**Example:**

int myArray[] = {1, 2, 3, 4, 5};

myArray[1] = 5;

*// change the value 2 to 5 at index 1*

Let's understand clearly with a Java program.

public class Demo {

public static void main(String[] args) {

*// declaring and initializing an array*

int[] myArray = { 2, 3, 4, 5, 6 };

*// print array element at index 0*

System.out.println("Before change myArray[0]: " + myArray[0]);

*// changing the array element*

myArray[0] = 1;

*// print array element at index 0*

System.out.println("After change myArray[0]: " + myArray[0]);

}

}

**Output:**

Before change myArray[0]: 2

After change myArray[0]: 1

We have created an int type array in the above code example by initializing five values. After that, we re-assigned the array element at index [0] from 2 to 1 just by using an array-defined index.

**Looping Through Array Elements**

For performing any operation or displaying any elements, we used an index. But by using an index, we can only implement one operation at a time. If we are required to modify all elements in an array, we must write an operation for each index.

To overcome this situation, we have a looping feature in Java that follows the principle **write once and execute multiple times.**

There are two ways to loop through the array elements.

1. **Using for Loop -** In this way of looping through the array elements, we need the array's length first. This can be done by java in-built property length. If we start the for loop from 0, then the total iteration will be the array's length - 1, and if we start the for loop from 1, then the total iteration will be the length of an array.

As shown in the above image, there is a total of 9 value in the array, and we start looping from 0 to 8.

**Example:**

public class Demo {

public static void main (String[] args) {

*// declaring and initializing an array*

String strArray[] = {"Python", "Java", "C++", "C", "PHP"};

*// Find the length of an array*

int lengthOfArray = strArray.length;

*// using for loop*

for(int i=0;i<lengthOfArray;i++) {

System.out.println(strArray[i]);

}

}

}

**Output:**

Python

Java

C++

C

PHP

In the above code, we have just used a for loop to go through the array element. For the number of iterations, we find the length of the array using the array length property.

1. **Using for-each loop -** In this way, we use a for loop, but we do not need the length of the array. The for-each method is much easier to loop through the array element than looping for-loop. While using the for-each loop, we need to define a variable of the same type as the data type of the defined array. If not done, the compiler will throw compile time error saying- incompatible types: String cannot be converted to YourDefinedDataType

**Example:**

public class Demo {

public static void main (String[] args) {

*// declaring and initializing an array*

String strArray[] = {"Python", "Java", "C++", "C", "PHP"};

*// using for-each loop*

for(String i : strArray) {

System.out.println(i);

}

}

}

**Output:**

Python

Java

C++

C

PHP

The above for-each code can be read - *for each String element in strArray, print the value of i*.

Apart from the above two looping methods, we can also use a while loop for traversing.

**Types of Array in Java**

In Java, there are two types of arrays:

1. Single-Dimensional Array
2. Multi-Dimensional Array

**1. Single Dimensional Array**

An array that has **only one subscript** or one dimension is known as a single-dimensional array. It is just a list of the same data type variables.

One dimensional array can be of either one row and multiple columns or multiple rows and one column. For instance, a student's marks in five subjects indicate a single-dimensional array.  
**Example:**

int marks[] = {56, 98, 77, 89, 99};

During the execution of the above line, JVM (Java Virtual Machine) will create five blocks at five different memory locations representing marks[0], marks[1], marks[2], marks[3], and marks[4].

We can also define a one-dimensional array by first declaring an array and then performing memory allocation using the new keyword.  
**Example:**

int marks[]; *// declared an array*

marks = new int[5]; *// allocating memory*

*// initializing array using index*

marks[0] = 56;

marks[1] = 98;

marks[2] = 77;

marks[3] = 89;

marks[4] = 99;

Let's look at how we can implement a single-dimensional array in Java.

public class Demo {

public static void main (String[] args) {

*// declaring and initializing an array*

String strArray[] = {"Python", "Java", "C++", "C", "PHP"};

*// using a for-each loop for printing the array*

for(String i : strArray) {

System.out.print(i + " ");

}

*// find the length of an array*

System.out.println("\nLength of array: "+strArray.length);

}

}

**Output:**

Python Java C++ C PHP

Length of array: 5

In the above example code, we created a single-dimensional array called strArray, which has only one row and multiple columns.

**2. Multi-Dimensional Array**

In not all cases, we implement a single-dimensional array. Sometimes, we are required to create an array within an array.

Suppose we need to write a program that stores five different subjects and the marks of sixty students. In this case, we just need to implement 60 one-dimensional arrays with a length of five and then implement all these arrays in one array. Hence, we can make the 2D array with five columns and 60 rows.

A multi-dimensional array is just an array of arrays that represents multiple rows and columns. These arrays include 2D, 3D, and nD types. 2D data like tables and matrices can be represented using this type of array.

To implement a multi-dimensional array, one needs to add two square brackets instead of one.  
**Example:**

int marks[][] = {

{77,85,68,99,87},

{98,56,79,90,92},

{78,88,56,70,99}

};

OR

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

*// both will create a 2D array with 3 rows and 5 columns.*

If we want to change any specific value, we need to use two square brackets, the first representing the rows and the second representing the columns.

marks[1][2] = 84 *// change value at 1st row and 2nd column from 79 to 84*

*//array will look like this -*

77,85,68,99,87

98,56,84,90,92

78,88,56,70,99

Now let's understand a multi-dimensional array with a famous Matrix multiplication example.

public class Demo {

public static void main (String[] args) {

*// declaring and initializing arrays*

int arr1[][] = {{1,2,3},{4,5,6},{7,8,9}};

int arr2[][] = {{2,2,2},{2,2,2},{2,2,2}};

*// Printing Array1 in matrix format*

System.out.println("Array1 -");

for(int i=0;i<3;i++) { *// for-loop for number of rows*

for(int j=0;j<3;j++) { *// for-loop for number of columns*

System.out.print(arr1[i][j] + " ");

}

System.out.println();

}

*// Printing Array2 in matrix format*

System.out.println("Array2 -");

for(int i=0;i<3;i++) { *// for-loop for number of rows*

for(int j=0;j<3;j++) { *// for-loop for number of columns*

System.out.print(arr2[i][j] + " ");

}

System.out.println();

}

*// creating another array with the same dimensions to store the result*

int arr3[][] = new int[3][3];

*// Multiplying arr1 and arr2, storing results in arr3*

System.out.println("Multiplication of Array1 and Array2 - ");

for(int i=0;i<arr1.length;i++) {

for(int j=0;j<arr2.length;j++) {

arr3[i][j] = 0;

for(int k=0;k<arr3.length;k++) {

arr3[i][j] += arr1[i][k] \* arr2[k][j];

}

System.out.print(arr3[i][j] + " ");

}

System.out.println();

}

}

}

**Output:**

Array1 -

1 2 3

4 5 6

7 8 9

Array2 -

2 2 2

2 2 2

2 2 2

Multiplication of Array1 and Array2 -

12 12 12

30 30 30

48 48 48

Matrix multiplication is a very famous example of a 2D array. In the above code, we declared two arrays of 3x3 dimensions. Then we created one more array with the same dimensions to store the result. Later we implemented a logic of matrix multiplication.

**Arrays of Objects**

As the name suggests, an array of objects is nothing but a list of objects stored in the array. Notice that it does not store objects in an array but stores the reference variable of that object. The syntax will be the same as above.

**Example:**

Student studentObj[] = new Student[3];

After the above statement executes, it will create an array of objects of the Student class with a length of 3 studentObj references. For each object reference, we need to implement an object using new.

Let’s understand it more clearly with a Java program.

class Student {

Student(int id, String name) {

System.out.println("Student ID is "+ id + " and name is "+ name );

}

}

public class Test {

public static void main (String[] args) {

*// declaring an array of Object*

Student obj[] = new Student[3];

obj[0] = new Student(1,"Bharat");

obj[1] = new Student(5,"Vivaan");

obj[2] = new Student(6,"Smith");

}

}

**Output:**

Student ID is 1 and name is Bharat

Student ID is 5 and name is Vivaan

Student ID is 6 and name is Smith

As you can see in the above code, we have first declared the array of objects with a length of three, which create three object reference variable obj[0], obj[1], and obj[2]. And then initialized each object reference by using a new keyword.

**Passing Arrays to Methods**

Likewise, we pass variables as a parameter and can also pass the array to methods. While passing, we do not require adding a square bracket with an array name, but we do require that in formal parameters.

Passing an array while calling methods is not required to mention the size of the array. If you do so, then it will throw the compile-time error.

Note that it is not required to use the same name of the array variable in the parameter of a function as the array name. We can use any name we want. But make sure to define the parameter variable with the same data type defined in an array in the main function, or else you will see a compilation error.

Let's understand it more clearly with a small Java program.

public class Test {

*// defined function to find max number from an array*

public static int findMax(int[] arr) {

int max = arr[0];

for(int i=1;i<arr.length;i++) {

if(max<arr[i]) {

max = arr[i];

}

}

return max;

}

public static void main (String[] args) {

*// declaring and initializing an array*

int myArray[] = {45,33,98,65,76,43,99,23,68};

*// call function by passing array in it*

int maxNumber = findMax(myArray);

System.out.println("The max number of array is " + maxNumber);

}

}

**Output:**

The max number of array is 99

In the above code, we have created a static function in which we passed the array myArray from the main function. Notice that while passing an array, it is not required to use [] brackets, but in function parameters, we need to use [] brackets to define the array.

**Returning Arrays from Methods**

Sometimes, we need to pass an array to a method to perform some operations and want that modified array as a return value of the function.

We already learned how we could pass arrays in methods. Let's now see how to send arrays back from the function using the return keyword.

Let's understand it more clearly with a small Java program.

public class Test {

*// defined function to find max number from an array*

public static int[] doMultiplication(int[] arr) {

for(int i=0;i<arr.length;i++) {

arr[i] = arr[i] \* 2;

}

return arr;

}

public static void main (String[] args) {

*// declaring and initializing an array*

int myArray[] = {1,2,3,4,5};

*// call function by passing array in it*

int[] multiplyArr = doMultiplication(myArray);

*// print array*

System.out.println("Array multiply by 2 is - " );

for(int i : multiplyArr) {

System.out.print(i + " ");

}

}

}

**Output:**

Array multiply by 2 is -

2 4 6 8 10

Like we defined the data type of a function according to the return type in an array, we also need to define a function with an array data type. We have done the same in the above code, defined function doMultiplication() of type int[], and stored the result of this method in int type array multiplyArr.

**Anonymous Array in Java**

As its name suggests, arrays having no name are called Anonymous arrays in java. This type of array is used only when we require an array instantly. We can also pass this array to any method without any reference variable.

As anonymous arrays do not have reference variables, we can not access its element, though we can use it entirely to meet our logic.

The primary reason to create an anonymous array is to implement and use an array instantly.

**Syntax:**

new datatype[] {values separated by comma}

**Example:**

*//One-dimensional integer type anonymous array*

new int[] {2,4,6,8};

*// Multi-dimensional integer type anonymous array*

new int[][] { {1,3,5}, {2,4,6} };

**Note:** In an anonymous array, we do not need to mention the size inside [] but the number of values we put inside {} becomes the size of the array.

**ArrayIndexOutOfBoundsException**

Till now, we saw how to access a specific element of an array and loop through all the array elements. But what if, by mistake, we try to access an index value that is either greater than the array length or negative?

In this situation, JVM throws the runtime exception of **ArrayIndexOutOfBoundsException**, which points to an array accessed by an illegal index.

If in the above example we try to access currencies[5] then the system will give us ArrayIndexOutOfBoundsException.

**Example:**

public class Test {

public static void main (String[] args) {

*// declaring and initializing an array*

int myArray[] = new int[5];

myArray[0] = 3;

myArray[2] = 6;

myArray[4] = 9;

myArray[6] = 12; *// index 6 not exists*

for(int i : myArray) {

System.out.println(i);

}

}

}

**Output:**

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 6 out of bounds for length 5

at Test.main(Test.java:9)

In the above code, we can see that all work fine till initializing index [4]. But, when we tried to initialize the array value at index [6], the compiler threw an exception, because we only have the array with a length of five.

**Cloning of Arrays in Java**

Cloning an array in Java is nothing but creating a new array, where data is copied from the existing one using the clone() property.

**Syntax:**

datatype Array2[] = Array1.clone();

**Example:**

public class Test{

public static void main (String[] args) {

*// declaring and initializing an array*

int myArray[] = {1,2,3,4,5};

*// clone myArray by using clone property*

int cloneArray[] = myArray.clone();

*// checking whether both arrays are the same or not*

System.out.println(myArray == cloneArray);

}

}

**Output:**

false

When we are cloning on the multi-dimensional array, a **shallow copy** is performed by creating a new single array with each element referring to an original array.

**Example:**

public class Test {

public static void main (String[] args) {

*// declaring and initializing an array*

int myArray[][] = { {1,2,3}, {4,5,6}};

*// clone myArray by using clone property*

int cloneArray[][] = myArray.clone();

*// checking whether both arrays are the same or not*

System.out.println(myArray == cloneArray);

*// checking whether both arrays have the same elements*

System.out.println(myArray[1] == cloneArray[1]);

}

}

**Output:**

false

true

As we can see, we cloned the 2D array cloneArray from myArray. Although both arrays are not the same, their elements point to the same value.