**Arrays in Java**

An array is a group of like-typed variables that are referred to by a common name. Arrays in Java work differently than they do in C/C++. Following are some important point about Java arrays.

In Java all arrays are dynamically allocated.

Since arrays are objects in Java, we can find their length using member length. This is different from C/C++ where we find length using sizeof.

A Java array variable can also be declared like other variables with [] after the data type.

The variables in the array are ordered and each have an index beginning from 0.

Java array can be also be used as a static field, a local variable or a method parameter.

The size of an array must be specified by an int value and not long or short.

The direct superclass of an array type is Object.

Array can contains primitives data types as well as objects of a class depending on the definition of array. In case of primitives data types, the actual values are stored in contiguous memory locations. In case of objects of a class, the actual objects are stored in heap segment.

**One-Dimensional Arrays :**

The general form of a one-dimensional array **declaration** is

type var-name[];

OR

type[] var-name;

An array declaration has two components: the type and the name. type declares the element type of the array. The element type determines the data type of each element that comprises the array. Like array of int type, we can also create an array of other primitive data types like char, float, double..etc or user defined data type(objects of a class).Thus, the element type for the array determines what type of data the array will hold.

**Instantiating an Array in Java**

When an array is declared, only a reference of array is created. To actually create or give memory to array, you create an array like this:The general form of new as it applies to one-dimensional arrays appears as follows:

var-name = new type [size];

Here, type specifies the type of data being allocated, size specifies the number of elements in the array, and var-name is the name of array variable that is linked to the array. That is, to use new to allocate an array, you must specify the type and number of elements to allocate.

Example:

int intArray[]; //declaring array

intArray = new int[20]; // allocating memory to array

OR

int[] intArray = new int[20]; // combining both statements in one

Note :

The elements in the array allocated by new will automatically be initialized to zero (for numeric types), false (for boolean), or null (for reference types).Refer Default array values in Java

Obtaining an array is a two-step process. First, you must declare a variable of the desired array type. Second, you must allocate the memory that will hold the array, using new, and assign it to the array variable. Thus, in Java all arrays are dynamically allocated.

**Array Literal**

In a situation, where the size of the array and variables of array are already known, array literals can be used.

int[] intArray = { 1,2,3,4,5,6,7,8,9,10 };

// Declaring array literal

The length of this array determines the length of the created array.

There is no need to write the new int[] part in the latest versions of Java

Accessing Java Array Elements using for Loop

Each element in the array is accessed via its index. The index begins with 0 and ends at (total array size)-1. All the elements of array can be accessed using Java for Loop.

// accessing the elements of the specified array

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

**System.out.println("Element of array "+ arr[i]);**

**// Java program to illustrate creating an array**

**// of integers,  puts some values in the array,**

**// and prints each value to standard output.**

class A

{

    public static void main (String[] args)

    {

      // declares an Array of integers.

      int[] arr;

      // allocating memory for 5 integers.

      arr = new int[5];

      // initialize the first elements of the array

      arr[0] = 10;

      // initialize the second elements of the array

      arr[1] = 20;

      //so on...

      arr[2] = 30;

      arr[3] = 40;

      arr[4] = 50;

      // accessing the elements of the specified array

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

         System.out.println("Element at index " + i +

                                      " : "+ arr[i]);

    }

}

OUTPUT:

Element at index 0 : 10

Element at index 1 : 20

Element at index 2 : 30

Element at index 3 : 40

Element at index 4 : 50

|  |
| --- |
| **// Java program to illustrate creating an array of**  **// objects**    class Student  {      public int roll\_no;      public String name;      Student(int roll\_no, String name)      {          this.roll\_no = roll\_no;          this.name = name;      }  }    // Elements of array are objects of a class Student.  public class GFG  {      public static void main (String[] args)      {          // declares an Array of Student          Student[] arr;            // allocating memory for 5 objects of type Student.          arr = new Student[5];            // initialize the first elements of the array          arr[0] = new Student(1,"aman");            // initialize the second elements of the array          arr[1] = new Student(2,"vaibhav");            // so on...          arr[2] = new Student(3,"shikar");          arr[3] = new Student(4,"dharmesh");          arr[4] = new Student(5,"mohit");            // accessing the elements of the specified array          for (int i = 0; i < arr.length; i++)              System.out.println("Element at " + i + " : " +                          arr[i].roll\_no +" "+ arr[i].name);      }  } |

Output:

Element at 0 : 1 aman

Element at 1 : 2 vaibhav

Element at 2 : 3 shikar

Element at 3 : 4 dharmesh

Element at 4 : 5 mohit

**What happens if we try to access element outside the array size?**

JVM throws ArrayIndexOutOfBoundsException to indicate that array has been accessed with an illegal index. The index is either negative or greater than or equal to size of array.

**Multidimensional Arrays**

A multidimensional array is an array containing one or more arrays.

To create a two-dimensional array, add each array within its own set of curly braces:

Example

int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };

myNumbers is now an array with two arrays as its elements.

To access the elements of the myNumbers array, specify two indexes: one for the array, and one for the element inside that array. This example accesses the third element (2) in the second array (1) of myNumbers:

Example

int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };

int x = myNumbers[1][2];

System.out.println(x); // Outputs 7

We can also use a for loop inside another for loop to get the elements of a two-dimensional array (we still have to point to the two indexes):

**Example**

public class MyClass {

public static void main(String[] args) {

int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };

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

for(int j = 0; j < myNumbers[i].length; ++j) {

System.out.println(myNumbers[i][j]);

}

}

}

}

**Passing Arrays to Methods**

Like variables, we can also pass arrays to methods.For example, below program pass array to method sum for calculating sum of array’s values.

class Test

{

// Driver method

public static void main(String args[])

{

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

// passing array to method

sum(arr);

}

public static void sum(int[] arr)

{

// getting sum of array values

int sum = 0;

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

sum+=arr[i];

System.out.println("sum of array values : " + sum);

}

}

Output :

sum of array values : 15

**Returning Arrays from Methods**

As usual, a method can also return an array.

class Test

{

// Driver method

public static void main(String args[])

{

int arr[] = m1();

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

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

}

public static int[] m1()

{

// returning array

return new int[]{1,2,3};

}

}

Output:

1 2 3