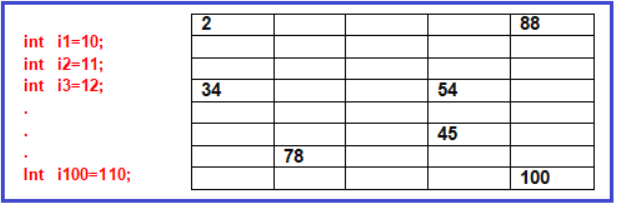
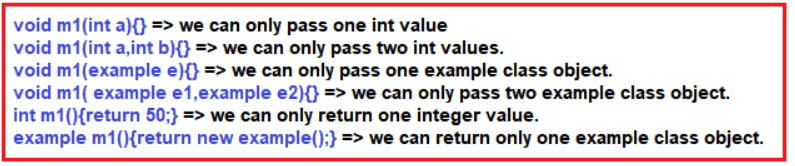
**Problem1:**

For example, if we want to store multiple values, say 1 to 100, we must create 100 variables, and those variables are created at different locations as shown in the below image. Hence it takes more time to return values.



**Problrm2:**

Also using primitive data type variables we cannot pass multiple values or objects to a method as an argument and also we cannot return multiple values and objects from a method at a time. For a better understanding please have a look at the below image.



**How do we overcome the above two problems???**

To solve the above two problems, we must group all values and objects to send them as a single unit from one application to another application as a method argument or return type. To group them as a single unit we must store them in contiguous memory locations. This can be possible by using a reference data types array.

In java, the array is a reference data type. It is used to store a fixed number of multiple values and objects of the same type in contiguous memory locations.

**Note:** Like other data types array is not a keyword rather it is a concept. It creates continuous memory locations using other primitive or reference types.

**Array Limitation:** Its size is fixed, which means we cannot increase or decrease its size after its creation.

**Features of Array**

1. Arrays are objects
2. They can even hold the reference variables of other objects
3. They are created during runtime
4. They are dynamic, created on the heap
5. The Array length is fixed

**Types of Arrays in Java:**

We have the following two types of Arrays:

1. Single Dimensional Array
2. Multi-Dimensional Array

**Single Dimensional Arrays**

Single Dimensional Arrays, means we are storing multiple values in one dimensional that is either horizontally or vertically.

**Declaration of Single Dimensional Arrays**

Declaration of an array means we are deciding what is the name of the array and what type of values we are storing. The syntax is given below.

Declaration of Single Dimensional Arrays

**Here**

1. Accessibility modifiers can be public, private, or protected.
2. The non-accessibility modifier can be static.
3. The data type can be any primitive data type or class type
4. Array name must be any java valid identifier
5. The array must be represented using []

**For example:**

**public static int[] intArr; //Here, int is primitive data type**  
**public static Employee[] empArr; //Employee is a class type or you can say reference type.**

**Rules for declaring Arrays in Java:**

You can place the pair of square brackets [] after the variable name, before the variable name, after the data type but not before the data type. For a better understanding, please have a look at the below image.



Like C or C++, in java, we cannot specify the array size in the declaration part. It leads to a compile-time error. For example:

**int[5] i; CE: illegal start of expression.**  
**int[] i; //Allowed**

**Creation of Single Dimensional Array in Java**

The creation of Single Dimensional array means providing the memory so that we can use it in the application. In two ways we can create an array in Java.

1. **Array Creation with values**
2. **Array Creation without Values**

**Array Creation with Values:**

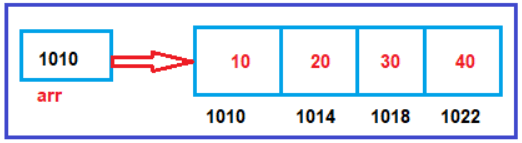
If you want to initialize an array with your own values then you have to use this concept. The syntax is given below.

Array Creation with Values

Creating an array in a single line means we are declaring, creating, and initializing an array at a time. The size of the array will be decided based on the number of values specified in between {}.

**Example Primitive Array: int arr[] = {10, 20, 30, 40};**

In this array object creation, the array contains 4 continuous memory locations with some starting base address say 1010, and that address is stored in the ‘arr’ variable as shown in the below image.

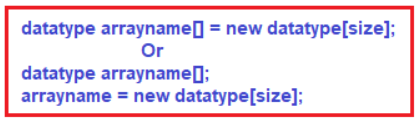


**Rules for creating an array in a single line**

1. While creating an array in a single line we must not specify the size, but the size of the array will be decided based on the number of values specified in between {}.
2. A list of values must be specified by using {}.

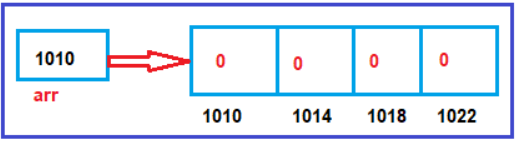
**Array Creation without values:**

**Syntax:**



**Example:** **int arr[] = new int[4];**

From the above statement array object is created with four int type variables, all locations are initialized with the default value zero because the array object is created with primitive data type integer as shown in the below image.



**Rules for Creating an Array in Java:**

1. While creating an array specifying array dimension or size is mandatory
2. Array dimension or size must be byte/ short/ int/ char but not long, float, double, or Boolean.
3. Array dimension or size must not be negative but if we write size as a negative value then the code is valid and compiled successfully but at run time it will throw a runtime exception saying NegativeArraySizeException.

**Accessing elements of an Array in Java**

To access the elements of an array we have to use a concept called indexes (sub-script reference). In Java, always array indexes start from 0 to size -1. While accessing if we write array index <0 or >=size then it will throw a run time exception saying ArrayIndexOutOfBoundsException.

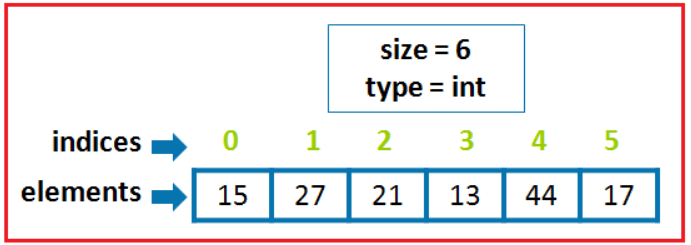
**What happens if we try to access an element outside the array size?**

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

**Example:**  
**int arr[] = new int[10];**  
**System.out.println(arr[0]); //0**  
**System.out.println(arr[5]); //0**  
**System.out.println(arr[12]); //ArrayIndexOutOfBoundException**  
**System.out.println(arr[-7]); //ArrayIndexOutOfBoundException**

In Java, arrays are maintained like an object. To create an array we have to use new operator. When an array is created the memory will be allocated in heap memory and each location of an array will get memory and initialized with default values.

**Example:** **int arr[] = new int[6]; Or** **int arr[] = {15, 27, 21, 13, 44, 17};**



In arrays all elements are stored in continuous memory locations and address of each location is decided based on the size of the data type and index like follows,

**Address of 1st Element** = startingaddress + index\*datatypesize  
= 500+0\*4  
= 500  
**Address of 2nd Element** = starting address + index\*datatype size  
= 500+1\*4  
= 504  
So on……….

**Example to demonstrate Single Dimensional Arrays in Java**

**package** *Demo*;

**public** **class** ArrayDemo

**{**

**public** **static** **void** main**(String[]** args**)** **{**

// declares an array of integers

**int[]** anArray;

// allocates memory for 5 integers

anArray = new **int[**5**]**;

// initialize first element

anArray**[**0**]** = 100;

// initialize second element

anArray**[**1**]** = 200;

// and so forth

anArray**[**2**]** = 300;

anArray**[**3**]** = 400;

anArray**[**4**]** = 500;

System.out.println**(**"Element at index 0: " + anArray**[**0**])**;

System.out.println**(**"Element at index 1: " + anArray**[**1**])**;

System.out.println**(**"Element at index 2: " + anArray**[**2**])**;

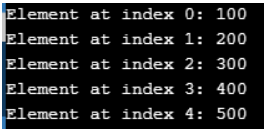
System.out.println**(**"Element at index 3: " + anArray**[**3**])**;

System.out.println**(**"Element at index 4: " + anArray**[**4**])**;

**}**

**}**

**Output:**



**Example to demonstrate an array in a single line**

**package** *Demo*;

**public** **class** SingleLineArray

**{**

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

**{**

**double** arr**[]** = **{** 10, 20, 56.4, 23.34, 12.67, 40 **}**;

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

**{**

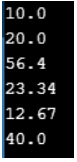
System.out.println **(**arr**[**i**]** + " "**)**;

**}**

**}**

**}**

**Output:**



**Note:**In Java, arrays are available as an object and there will be a predefined variable called “length” is available for every array which indicates the count of the number of elements available in the array (size of the array).

**Accessing Java array Elements using forEach Loop**

The for-each loop is a special loop introduced in Java 1.5 Version which is used to access the elements of an array or any collection object. foreach loop is not a normal loop that is generally used to repeat the group of statements. To write for each loop again we have to write for keyword. The syntax is given below.

**for(declaration : arrayobj / collectionobj){**  
**//statements;**  
**}**

**Rules:**

1. for each loop must contain 2 sections separated by “:”
2. The first section must be a declaration of a variable to hold each value of a particular array or collection.
3. The second section must be an array or collection object but it must not be an expression or value.

**Example to demonstrate the use of forEach Loop in an Array:**

**package** *Demo*;

**public** **class** ForEachDemo

**{**

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

**{**

**double** arr**[]** = **{** 1.9, 2.9, 3.4, 3.5 **}**;

**for** **(double** v:arr**)**

**{**

System.out.println **(**v + " "**)**;

**}**

**}**

**}**

**Output:**

Program to demonstrate the use of forEach Loop in an Array

**Arrays of Objects or Reference Array in Java:**

Let us understand this with an example. Let’s say we have the following Sample class.

**class Sample{**  
**int x=10;**  
**int y=20;**  
**}**

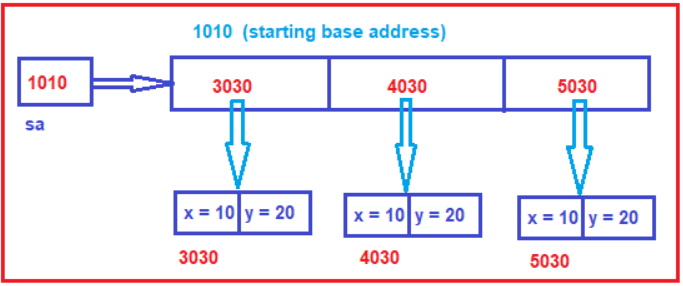
An array of objects is created just like an array of primitive type data items in the following way.

**Sample[] arr = new Sample[3];** //Sample is a user-defined class

The Sample Array contains 3 memory spaces each of the size of the Sample class in which the address of 3 sample objects can be stored. The Sample objects have to be instantiated using the constructor of the Sample class and their references should be assigned to the array elements.

**Sample[] sa={new Sample(), new Sample(), new Sample()};**

In the above array object creation, the array contains 3 elements with continuous memory locations with some starting base address assume 1010, and that base address is stored in sa variable as shown in the below diagram



**What is the difference between creating an array object with primitive types and referenced types???**

1. If we create array objects with primitive types, all its memory locations are of primitive type variables, so values are stored directly in those locations.
2. If we create an array object with a referenced type, all its memory locations are of referenced type variables, so object reference is stored in those memory locations.

**Java Program to illustrate creating an array of object**

**package** *Demo*;

**class** Student123

**{**

**public** **int** roll\_no;

**public** **String** name;

Student123**(int** roll\_no, **String** name**)**

**{**

this.roll\_no = roll\_no;

this.name = name;

**}**

**}**

// Elements of array are objects of a class Student.

**public** **class** ArrayOfObject

**{**

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

**{**

// declares an Array of integers.

Student123**[]** arr;

// allocating memory for 5 objects of type Student.

arr = new Student123**[**5**]**;

// initialize the first elements of the array

arr**[**0**]** = new Student123**(**1,"aman"**)**;

// initialize the second elements of the array

arr**[**1**]** = new Student123**(**2,"vaibhav"**)**;

// so on...

arr**[**2**]** = new Student123**(**3,"shikar"**)**;

arr**[**3**]** = new Student123**(**4,"dharmesh"**)**;

arr**[**4**]** = new Student123**(**5,"mohit"**)**;

// accessing the elements of the specified array

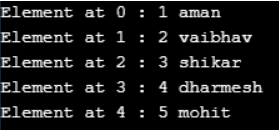
**for** **(int** i = 0; i **<** arr.length; i++**)**

System.out.println**(**"Element at " + i + " : " + arr**[**i**]**.roll\_no +" "+ arr**[**i**]**.name**)**;

**}**

**}**

**Output:**



**Note:**You can use the built-in length property to determine the size of an array. The following code prints the array’s size to standard output:

**System.out.println(anArray.length);**

**Advantages of Array in Java:**

1. It makes the code optimized, we can retrieve or sort the data efficiently.
2. We can get any data located at an index position.

**Disadvantages of Array in Java:**

1. We can store only the fixed size of elements in the array. It doesn’t grow its size at runtime.
2. To solve this problem, the collection framework is used in Java which grows automatically.

**Multi-Dimensional Arrays in Java:**

Multi-Dimensional Arrays means storing multiple values in more than one dimension. The simplest form of Multi-Dimensional Arrays is 2 Dimensional Array.

**Declaration of Two Dimensional Array**

**Syntax:**

**Datatype arrayname[][];**  
**Example:**  
**int arr[][];**  
**int[][] arr;**  
**int [][]arr;**

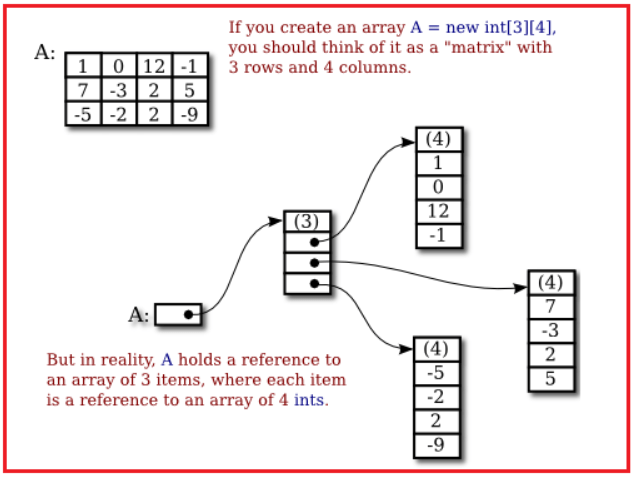
**Creation of Two Dimensional Array in Java**

**Syntax : datatype arrayname[][] = new datatype[size][size];**  
**Example : int arr[][] = new int[][];**

**Rules for Two Dimensional Array**

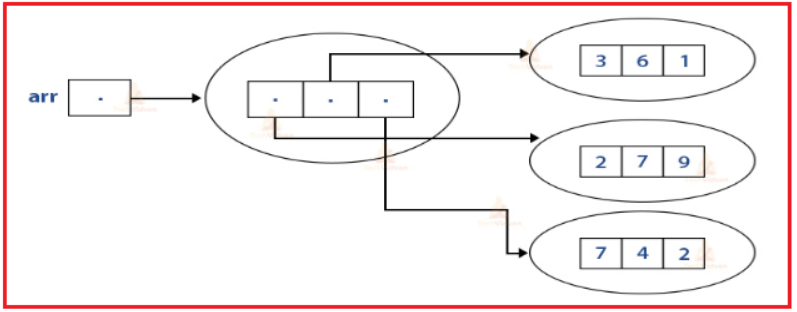
1. When we create a Two Dimensional array specifying the size for the first square bracket is mandatory and for next square bracket, it is optional.
2. Here each square bracket will indicate one single dimensional array.
3. Here multi-dimensional arrays are stored in multiple levels where all top levels of arrays are containing addresses of the next level of arrays and only the last level of arrays contains values.

**Example:** **int A[][] = new int[3][4];**



**Note:**In the last level of arrays we may have arrays with an equal number of elements or a non-equal number of elements. We can also write more than 2 dimensions for arrays hence multidimensional arrays are also called an array of arrays.

**Example: int arr[][][] = new int[3][2][4]**



**Sample Program to demonstrate the use of Multi-Dimensional Array in Java**

**package** *Demo*;

**public** **class** MultidimArrayDemo **{**

**public** **static** **void** main**(String[]** args**)** **{**

**String[][]** names = **{**

**{**"Mr. ", "Mrs. ", "Ms. "**}**,

**{**"Smith", "Jones"**}**

**}**;

// Mr. Smith

System.out.println**(**names**[**0**][**0**]** + names**[**1**][**0**])**;

// Ms. Jones

System.out.println**(**names**[**0**][**2**]** + names**[**1**][**1**])**;

**}**

**}**

**Output :**



**Copying Arrays in Java**

The System class has an arraycopy method that you can use to efficiently copy data from one array into another. The syntax is given below.

Copying Arrays in Java

The two Object arguments specify the array to copy from and the array to copy to. The three int arguments specify the starting position in the source array, the starting position in the destination array, and the number of array elements to copy.

**Sample Program to demonstrate the use of arraycopy method in Java**

**package** *Demo*;

**public** **class** ArrayCopyDemo **{**

**public** **static** **void** main**(String[]** args**)** **{**

**char[]** copyFrom = **{** 'd', 'e', 'c', 'a', 'f', 'f', 'e',

'i', 'n', 'a', 't', 'e', 'd' **}**;

**char[]** copyTo = new **char[**7**]**;

System.arraycopy**(**copyFrom, 2, copyTo, 0, 7**)**;

System.out.println**(**new **String(**copyTo**))**;

**}**

**}**

**Output: caffein**

**Passing Arrays to Methods in Java**

You can also pass arrays to methods just as you can pass primitive type values to methods.

**Example:**

**public static void printArray(int[] array) {**  
**for (int i = 0; i < array.length; i++) {**  
**System.out.print(array[i] + ” “);**  
**}**  
**}**

In the above example, printArray method displays the elements in an **int** array. You can invoke it by passing an array.

**Example: printArray(new int[]{3, 1, 2, 6, 4, 2});**

**Sample Program to pass an array to a method in java**

**public** **class** ArrayDemo

**{**

**public** **static** **void** updateParam **(double[]**x**)**

**{**

x**[**0**]** = 9999; // Update one of the array element

**}**

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

**{**

**double[]** a = **{** 2.3, 3.4, 4.5 **}**;

System.out.println **(**"Array before calling updateParam:"**)**;

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

System.out.println **(**a**[**i**])**;

updateParam **(**a**)**; // Call updateParam

System.out.println **(**"Array AFTER calling updateParam:"**)**;

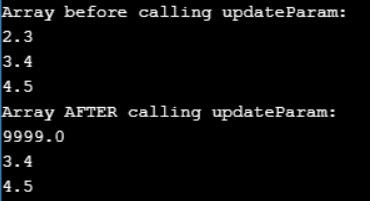
**for** **(int** i = 0; i **<** a.length; i++**)**

System.out.println **(**a**[**i**])**;

**}**

**}**

**Output:**



**Returning an Array from a Method in Java**

A method may also return an array. An example is given below.

In the above example, the method returns an array that is the reversal of another array.

**Sample Program to return an array from a method in java:**

**import** *java.util.Arrays*;

**import** *java.util.Scanner*;

**public** **class** ArrayDemo

**{**

**public** **int[]** createArray**()** **{**

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

System.out.println**(**"Enter the size of the array that is to be created:: "**)**;

**int** size = sc.nextInt**()**;

**int[]** myArray = new **int[**size**]**;

System.out.println**(**"Enter the elements of the array ::"**)**;

**for(int** i=0; i**<**size; i++**)** **{**

myArray**[**i**]** = sc.nextInt**()**;

**}**

**return** myArray;

**}**

**public** **static** **void** main**(String** args**[])** **{**

Main obj = new Main**()**;

**int** arr**[]** = obj.createArray**()**;

System.out.println**(**"Array created is :: "+Arrays.toString**(**arr**))**;

**}**

**}**

**Output**

