Array Concept in Java

Topics

1. Introduction
2. Array Declaration
3. Array creation
4. Array Initialization
5. All in one
6. Length vs length()

**Java array** is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

Advantages

* **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
* **Random access:** We can get any data located at an index position.

Disadvantages

* **Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.
* **Array are fixed in size.** And it can hold only homogenous data types element.

## **Single Dimensional Array in Java**

**Syntax to Declare an Array in Java**

1. dataType[] arr; (or)
2. dataType []arr; (or)
3. dataType arr[];

**Instantiation of an Array in Java**

1. arrayRefVar=**new** datatype[size];

// String[] dynamicSizeofArray ---> the best way to declare the array because named is clearly separated from the data type of array.

## **Declaring Array Variables**

To use an array in a program, you must declare a variable to reference the array, and you must specify the type of array the variable can reference. Here is the syntax for declaring an array variable −

### Syntax

dataType[] arrayRefVar; // preferred way.

or

dataType arrayRefVar[]; // works but not preferred way.

## **Creating Arrays**

You can create an array by using the new operator with the following syntax −

### Syntax

arrayRefVar = new dataType[arraySize];

The above statement does two things −

* It creates an array using new dataType[arraySize].
* It assigns the reference of the newly created array to the variable arrayRefVar.

Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement, as shown below −

dataType[] arrayRefVar = new dataType[arraySize];

Alternatively you can create arrays as follows −

dataType[] arrayRefVar = {value0, value1, ..., valuek};

## **Processing Arrays**

When processing array elements, we often use either **for** loop or **foreach** loop because all of the elements in an array are of the same type and the size of the array is known.

## **The foreach Loops**

JDK 1.5 introduced a new for loop known as foreach loop or enhanced for loop, which enables you to traverse the complete array sequentially without using an index variable.

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

// Print all the array elements

for (double element: myList) {

System.out.println(element);

}

## **Passing Arrays to Methods**

Just as you can pass primitive type values to methods, you can also pass arrays to methods. For example, the following method displays the elements in an **int** array −

### Example

public static void printArray(int[] array) {

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

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

}

}

You can invoke it by passing an array. For example, the following statement invokes the printArray method to display 3, 1, 2, 6, 4, and 2 −

### Example

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

## **Returning an Array from a Method**

A method may also return an array. For example, the following method returns an array that is the reversal of another array −

### Example

public static int[] reverse(int[] list) {

int[] result = new int[list.length];

for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {

result[j] = list[i];

}

return result;

}

**Question 2: Can you store String in an array of Integer in Java? compile time error or runtime exception?**[[answer](http://javarevisited.blogspot.co.uk/2013/11/java-array-101-for-programmers-and.html)]  
This is a tricky question. The answer is both yes and no. You cannot store a String in an array of primitive int, it will result in compile time error as shown below, but if you create an array of Object and assign String[] to it and then try to store Integer object on it.  
  
The compiler won't be able to detect that and it will throw ArrayStoreExcpetion at runtime.

Arrays class in java

java.util

## Class Arrays

* [java.lang.Object](https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html)
  + java.util.Arrays

public class ****Arrays****

extends [Object](https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html)

This class contains various methods for manipulating arrays (such as sorting and searching). This class also contains a static factory that allows arrays to be viewed as lists.

The methods in this class all throw a NullPointerException, if the specified array reference is null, except where noted.

This class is a member of the [Java Collections Framework](https://docs.oracle.com/javase/7/docs/technotes/guides/collections/index.html).

**Syntax to use Array:**

Arrays.<function name>;

**Need for the Java-Arrays Class:**  
There are often times when [**loops**](https://www.geeksforgeeks.org/loops-in-java/) are used to do some tasks on an array like:

* Fill an array with a particular value.
* Sort an Arrays.
* Search in an Arrays.
* And many more.

Arrays class provides several static methods that can be used to perform these tasks directly without the use of loops.

**Methods in Java Array:**

The Arrays class of the [java.util package](https://www.geeksforgeeks.org/java-util-package-java/) contains several static methods that can be used to fill, sort, search, etc in arrays. These are:

1. [**static <T> List<T> asList(T… a)**](https://www.geeksforgeeks.org/arrays-aslist-method-in-java-with-examples/): This method returns a fixed-size list backed by the specified Arrays.

       // Get the Array

        int intArr[] = { 10, 20, 15, 22, 35 };

        // To convert the elements as List

        System.out.println("Integer Array as List: "

                           + Arrays.asList(intArr));

    }

**public static <T> List<T> asList(T... a)**

Returns a fixed-size list backed by the specified array. (Changes to the returned list "write through" to the array.) This method acts as bridge between array-based and collection-based APIs, in combination with Collection.toArray(). The returned list is serializable and implements RandomAccess.

This method also provides a convenient way to create a fixed-size list initialized to contain several elements:

     List<String> stooges = Arrays.asList("Larry", "Moe", "Curly");

**Parameters:**

a - the array by which the list will be backed

**Returns:**

a list view of the specified array

**compareUnsigned(array 1, array 2)**: This method compares two arrays lexicographically, numerically treating elements as unsigned.

 int intArr[] = { 10, 20, 15, 22, 35 };

        // Get the second Arrays

        int intArr1[] = { 10, 15, 22 };

        // To compare both arrays

        System.out.println("Integer Arrays on comparison: "

                           + Arrays.compareUnsigned(intArr, intArr1));

    }

[**static <T> int binarySearch(T[] a, int fromIndex, int toIndex, T key, Comparator<T> c)**](https://www.geeksforgeeks.org/arrays-binarysearch-in-java-with-examples-set-2-search-in-subarray/): This method searches a range of the specified array for the specified object using the binary search algorithm.

 // Get the Array

        int intArr[] = { 10, 20, 15, 22, 35 };

        Arrays.sort(intArr);

        int intKey = 22;

        System.out.println(

            intKey

            + " found at index = "

            + Arrays

                  .binarySearch(intArr, 1, 3, intKey));

    }

**compare(array 1, array 2)**: This method compares two arrays passed as parameters lexicographically.

**compareUnsigned(array 1, array 2)**: This method compares two arrays lexicographically, numerically treating elements as unsigned.

[**copyOf(originalArray, newLength)**](https://www.geeksforgeeks.org/arrays-copyof-in-java-with-examples/): This method copies the specified array, truncating or padding with the default value (if necessary) so the copy has the specified length.

[**static boolean deepEquals(Object[] a1, Object[] a2)**](https://www.geeksforgeeks.org/java-util-arrays-deepequals-java/): This method returns true if the two specified arrays are deeply equal to one another.

  int intArr[][] = { { 10, 20, 15, 22, 35 } };

        // Get the second Arrays

        int intArr1[][] = { { 10, 15, 22 } };

        // To compare both arrays

        System.out.println("Integer Arrays on comparison: "

                           + Arrays.deepEquals(intArr, intArr1));

[**equals(array1, array2)**](https://www.geeksforgeeks.org/java-util-arrays-equals-java-examples/): This method checks if both the arrays are equal or not.

[**fill(originalArray, fillValue)**](https://www.geeksforgeeks.org/arrays-fill-java-examples/): This method assigns this fillValue to each index of this Arrays.