

1. **Arrays in Java**

**1.1. Definition**

An array in Java provides represent a data structure that stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data or variables of the same type.

For instance, instead of declaring individual variables, such as var\_01, var\_02, ..., and var\_10, we can declare one array variable such as "myNumbers" and use myNumbers [0], myNumbers [1], and ..., myNumbers [10] to represent individual variables.

**1.2. Declaring Arrays**

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.

**Note** − The style dataType[] arrayRefVar is preferred. The style dataType arrayRefVar[] comes from the C/C++ language and was adopted in Java to accommodate C/C++ programmers.

**Example**

The following code snippets are examples of this syntax −

double[] myList; // preferred way.

or

double myList[]; // works but not preferred way.

**1.3. 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};

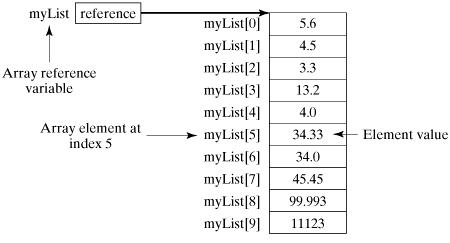
The array elements are accessed through the index. Array indices are 0-based; that is, they start from 0 to arrayRefVar.length-1.

**Example**

Following statement declares an array variable, myList, creates an array of 10 elements of double type and assigns its reference to myList.

double[] myList = new double[10];

Following picture represents array myList. Here, myList holds ten double values and the indices are from 0 to 9.



**1.4. Iterating Arrays**

Each variable in an Java array is also called an "element". Thus, the example shown earlier created an array with space for 10 elements, and each element is a variable of type int.

Each element in the array has an index (a number). You can access each element in the array via its index. Here is an example:

intArray[0] = 0;

int firstInt = intArray[0];

This example first sets the value of the element (int) with index 0, and second it reads the value of the element with index 0 into an int variable.

You can use the elements in a Java array just like if they were ordinary variables. You can read their value, assign values to them, use the elements in calculations and pass specific elements as parameters to method calls.

The indexes of elements in a Java array always start with 0 and continue to the number 1 below the size of the array. Thus, in the example above with an array with 10 elements the indexes go from 0 to 9.

## Array Length

You can access the length of an array via its length field. Here is an example:

int[] intArray = new int[10];

int arrayLength = **intArray.length;**

In this example the variable named arrayLength will contain the value 10 after the second line of code has been executed.

You can loop through all the elements of an array using the [**Java for loop**](http://tutorials.jenkov.com/java/for.html). Here is an example of iterating an array with a for loop in Java:

String[] stringArray = new String[10];

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

stringArray[i] = "String no " + i;

}

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

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

}

This example first creates an array of String references. When you first create an array of object references, each of the cells in the array points to null - no object.

The first of the two for loops iterate through the String array, creates a String and makes the cell reference that String.

The second of the two for loops iterate through the String array and prints out all of the strings that the cells reference.

If this had been an array of int (primitive values), it could have looked like this:

int[] intArray = new int[10];

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

intArray[i] = i;

}

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

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

}

The variable i is initialized to 0 and runs up until the length of the array minus 1. In this case, i takes the values 0 through 9, each time repeating the code inside the for loop one time, and for each iteration i has a different value.

You can also iterate an array using the "for-each" loop in Java. Here is how that looks:

int[] intArray = new int[10];

for(int theInt : intArray) {

System.out.println(theInt);

}

The for-each loop gives you access to each element in the array, one at a time, but gives you no information about the index of each element. Additionally, you only have access to the value. You cannot change the value of the element at that position. If you need that, use a normal for-loop as shown earlier.

For for-each loop also works with arrays of objects. Here is an example showing you how to iterate an array of String objects:

String[] stringArray = {"one", "two", "three"};

for(String theString : stringArray) {

System.out.println(theString);

}

**1.5. Multidimensional Arrays**

The examples shown above all created arrays with a single dimension, meaning elements with indexes going from 0 and up. It is, however, possible to create arrays where each element has two or more indexes which identify (locate) it in the array.

You create a multidimensional array in Java by appending one set of square brackets ([]) per dimension you want to add. Here is an example that creates a two-dimensional array:

int[][] intArray = new int[10][20];

This example creates a two-dimensional array of int elements. The array contains 10 elements in the first dimension, and 20 elements in the second dimension. In other words, this examples creates an array of arrays of int elements. The array of arrays has space for 10 int arrays, and each int array has space for 20 int elements.

You access the elements in a multidimensional array with one index per dimension. In the example above you would have to use two indexes. Here is an example:

int[][] intArray = new int[10][20];

intArray[0][2] = 129;

int oneInt = intArray[0][2];

The variable named oneInt will contain the value 129 after the last line of Java code has executed.

### Iterating Multidimensional Arrays

When you iterate a multidimensional array in Java you need to iterate each dimension of the array separately. Here is is how iterating a multidimensional looks in Java:

int[][] intArray = new int[10][20];

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

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

System.out.println("i: " + i + ", j: " + j);

}

}

**1.6. Inserting Elements Into an Array**

Sometimes you need to insert elements into a Java array somewhere. Here is how you insert a new value into an array in Java:

int[] ints = new int[20];

int insertIndex = 10;

int newValue = 123;

//move elements below insertion point.

for(int i=ints.length-1; i > insertIndex; i--){

ints[i] = ints[i-1];

}

//insert new value

ints[insertIndex] = newValue;

System.out.println(Arrays.toString(ints));

The example first creates an array. Then it defines an insert index and a new value to insert. Then all elements from the insertion index and to the end of the array are shifted one index down in the array. Note that this will shift the last value in the array out of the array (it will simply be deleted).

The above array insertion code could be embedded in a Java method. Here is how that could look:

public void insertIntoArray(

int[] array, int insertIndex, int newValue){

//move elements below insertion point.

for(int i=array.length-1; i > insertIndex; i--){

array[i] = array[i-1];

}

//insert new value

array[insertIndex] = newValue;

}

This method takes an int[] array as parameter as well as the index to insert the new value, and the new value. You can insert elements into an array by calling this method like this:

int[] ints = new int[20];

insertIntoArray(ints, 0, 10);

insertIntoArray(ints, 1, 23);

insertIntoArray(ints, 9, 67);

Of course, if the insertIntoArray() method is located in a different class than the above code, you would need an object of that class in order to be able to call the method. Or, if the insertIntoArray() method was static, you would need to put the class name and a dot in front of the method name.

**1.7. Removing Elements From an Array**

Sometimes you have want to remove an element from a Java array. Here is the code for removing an element from an array in Java:

int[] ints = new int[20];

ints[10] = 123;

int removeIndex = 10;

for(int i = removeIndex; i < ints.length -1; i++){

ints[i] = ints[i + 1];

}

This example first creates an int array. Then it sets the value of the element with index 10 to 123. Then the example removes the element with index 10. It removes the element by shifting all elements below index 10 one position up in the array. After the removal, the last element in the array will exist twice. Both in the last and second last element.

The above code could be embedded in a Java method. Here is how such an array removal Java method could look:

public void removeFromArray(

int[] array, int removeIndex){

for(int i = removeIndex; i < array.length -1; i++){

array[i] = array[i + 1];

}

}

This removeFromArray() method takes two parameters: The array to remove the element from, and the index of the element to remove.

Of course, if the removeFromArray() method is located in a different class than the above code, you would need an object of that class in order to be able to call the method. Or, if the removeFromArray() method was static, you would need to put the class name and a dot in front of the method name.

**1.8. Sorting Arrays**

You can sort the elements of an array using the Arrays.sort() method. Sorting the elements of an array rearranges the order of the elements according to their sort order. Here is an Arrays.sort() example:

int[] ints = new int[10];

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

ints[i] = 10 - i;

}

System.out.println(java.util.Arrays.toString(ints));

java.util.Arrays.sort(ints);

System.out.println(java.util.Arrays.toString(ints));

The first line declares and instantiates an array of int with a length of 10;

The for loop iterates over the array and inserts values into each element. The values inserted will go from 10 to 1 in descending order.

After the for loop the array is converted to a String using Arrays.toString() and printed out to the console (command line). At this point the output written to the console (the String version of the array) looks like this:

[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

The array is then sorted using Arrays.sort(). The elements will now be ordered in ascending order.

After sorting the array, it is again converted into a String and printed to the console. The output printed this time looks like this:

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

**2. Practice Arrays**

**2.1. Watch a video on Array methods**

**2.2. Quiz on Arrays**

**2.** **Polymorphism**