Java Copy Arrays

In this tutorial, you will learn about different ways you can use to copy arrays (both one dimensional and two-dimensional) in Java with the help of examples.

In Java, we can copy one array into another. There are several techniques you can use to copy arrays in Java.

---

1. Copying Arrays Using Assignment Operator

Let's take an example,

```
class Main {

    public static void main(String[] args) {


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

        int [] positiveNumbers = numbers;    // copying
arrays


        for (int number: positiveNumbers) {

            System.out.print(number + ", ");

        }
```

```
    }
}
```

Run Code

Output:

```
1, 2, 3, 4, 5, 6
```

In the above example, we have used the assignment operator (=) to copy an array named *numbers* to another array named *positiveNumbers*.

This technique is the easiest one and it works as well. However, there is a problem with this technique. If we change elements of one array, corresponding elements of the other arrays also change. For example,

```
class Main {

    public static void main(String[] args) {


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

        int [] positiveNumbers = numbers;    // copying arrays


        // change value of first array
```

```
        numbers[0] = -1;


        // printing the second array
        for (int number: positiveNumbers) {
            System.out.print(number + ", ");
        }
    }
}
```

Run Code

Output:

-1, 2, 3, 4, 5, 6

Here, we can see that we have changed one value of the *numbers* array. When we print the *positiveNumbers* array, we can see that the same value is also changed.

It's because both arrays refer to the same array object. This is because of the shallow copy. To learn more about shallow copy, visit shallow copy.

Now, to make new array objects while copying the arrays, we need deep copy rather than a shallow copy.

2. Using Looping Construct to Copy Arrays

Let's take an example:

```
import java.util.Arrays;


class Main {
    public static void main(String[] args) {


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


        // iterate and copy elements from source to
destination
        for (int i = 0; i < source.length; ++i) {
            destination[i] = source[i];
        }


         // converting array to string
```

```
        System.out.println(Arrays.toString(destination));
    }
}
```

Run Code

Output:

```
[1, 2, 3, 4, 5, 6]
```

In the above example, we have used the for loop to iterate through each element of the source array. In each iteration, we are copying elements from the *source* array to the *destination* array.

Here, the source and destination array refer to different objects (deep copy). Hence, if elements of one array are changed, corresponding elements of another array is unchanged.

Notice the statement,

```
System.out.println(Arrays.toString(destination));
```

Here, the toString() method is used to convert an array into a string. To learn more, visit the toString() method (official Java documentation).

---

3. Copying Arrays Using arraycopy() method

In Java, the System class contains a method named arraycopy() to copy arrays. This method is a better approach to copy arrays than the above two.

The arraycopy() method allows you to copy a specified portion of the source array to the destination array. For example,

arraycopy(Object src, int srcPos,Object dest, int destPos, int length)

Here,

- *src* - source array you want to copy
- *srcPos* - starting position (index) in the source array
- *dest* - destination array where elements will be copied from the source
- *destPos* - starting position (index) in the destination array
- *length* - number of elements to copy

Let's take an example:

```
// To use Arrays.toString() method

import java.util.Arrays;
```

```
class Main {
    public static void main(String[] args) {
        int[] n1 = {2, 3, 12, 4, 12, -2};

        int[] n3 = new int[5];

        // Creating n2 array of having length of n1 array
        int[] n2 = new int[n1.length];

        // copying entire n1 array to n2
        System.arraycopy(n1, 0, n2, 0, n1.length);
        System.out.println("n2 = " +
Arrays.toString(n2));

        // copying elements from index 2 on n1 array
        // copying element to index 1 of n3 array
        // 2 elements will be copied
        System.arraycopy(n1, 2, n3, 1, 2);
```

```
        System.out.println("n3 = " +
Arrays.toString(n3));
    }
}
```

Run Code

Output:

```
n2 = [2, 3, 12, 4, 12, -2]
n3 = [0, 12, 4, 0, 0]
```

In the above example, we have used the arraycopy() method,

- **System.arraycopy(n1, 0, n2, 0, n1.length) - entire elements from the *n1* array are copied to *n2* array**
- **System.arraycopy(n1, 2, n3, 1, 2) - *2* elements of the *n1* array starting from index *2* are copied to the index starting from *1* of the *n3* array**

As you can see, the default initial value of elements of an *int* type array is 0.

---

4. Copying Arrays Using copyOfRange() method

We can also use the copyOfRange() method defined in Java Arrays class to copy arrays. For example,

```
// To use toString() and copyOfRange() method

import java.util.Arrays;


class ArraysCopy {

    public static void main(String[] args) {


        int[] source = {2, 3, 12, 4, 12, -2};


        // copying entire source array to destination

        int[] destination1 = Arrays.copyOfRange(source,
0, source.length);

        System.out.println("destination1 = " +
Arrays.toString(destination1));


        // copying from index 2 to 5 (5 is not included)

        int[] destination2 = Arrays.copyOfRange(source,
2, 5);
```

```
        System.out.println("destination2 = " +
Arrays.toString(destination2));
    }
}
```

Run Code

Output

```
destination1 = [2, 3, 12, 4, 12, -2]
destination2 = [12, 4, 12]
```

In the above example, notice the line,

```
int[] destination1 = Arrays.copyOfRange(source, 0,
source.length);
```

Here, we can see that we are creating the *destination1* array and copying the *source* array to it at the same time. We are not creating the *destination1* array before calling the copyOfRange() method. To learn more about the method, visit Java copyOfRange.

---

5. Copying 2d Arrays Using Loop

Similar to the single-dimensional array, we can also copy the 2-dimensional array using the for loop. For example,

```
import java.util.Arrays;


class Main {

    public static void main(String[] args) {


        int[][] source = {
            {1, 2, 3, 4},
            {5, 6},
            {0, 2, 42, -4, 5}
            };


        int[][] destination = new int[source.length][];


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

```
            // allocating space for each row of destination
array
            destination[i] = new int[source[i].length];

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

        // displaying destination array

System.out.println(Arrays.deepToString(destination));

    }
}
```

Run Code

Output:

```
[[1, 2, 3, 4], [5, 6], [0, 2, 42, -4, 5]]
```

In the above program, notice the line,

System.out.println(Arrays.deepToString(destination);

Here, the deepToString() method is used to provide a better representation of the 2-dimensional array. To learn more, visit Java deepToString().

Copying 2d Arrays using arraycopy()

To make the above code more simpler, we can replace the inner loop with System.arraycopy() as in the case of a single-dimensional array. For example,

```
import java.util.Arrays;


class Main {

    public static void main(String[] args) {


        int[][] source = {

                {1, 2, 3, 4},

                {5, 6},

                {0, 2, 42, -4, 5}
```

```
        };

        int[][] destination = new int[source.length][];

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

            // allocating space for each row of destination
array
            destination[i] = new int[source[i].length];
            System.arraycopy(source[i], 0, destination[i], 0,
destination[i].length);
        }

        // displaying destination array

System.out.println(Arrays.deepToString(destination));
    }
}
```

Run Code

Output:

[[1, 2, 3, 4], [5, 6], [0, 2, 42, -4, 5]]

Here, we can see that we get the same output by replacing the inner for loop with the arraycopy() method.

Java Multidimensional Arrays

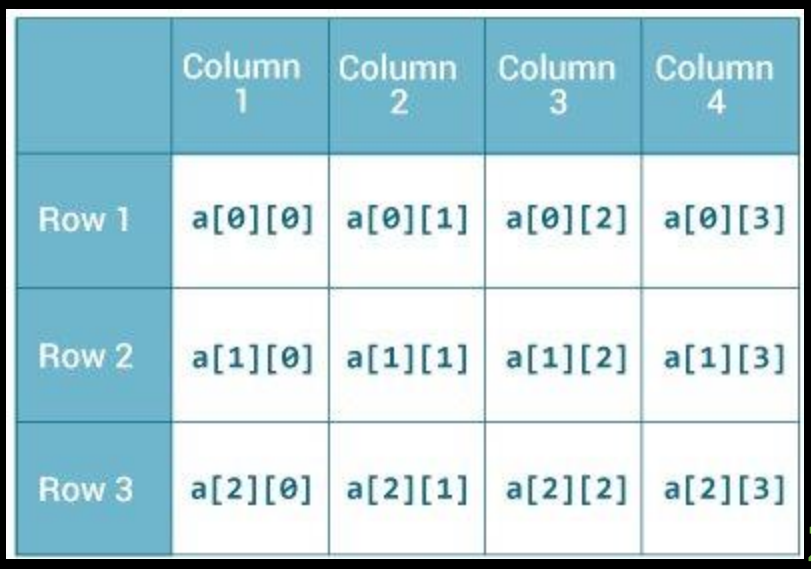
In this tutorial, we will learn about the Java multidimensional array using 2-dimensional arrays and 3-dimensional arrays with the help of examples.

Before we learn about the multidimensional array, make sure you know about Java array.

A multidimensional array is an array of arrays. Each element of a multidimensional array is an array itself. For example,

```
int[][] a = new int[3][4];
```

Here, we have created a multidimensional array named *a*. It is a 2-dimensional array, that can hold a maximum of 12 elements,



2-dimensional Array

Remember, Java uses zero-based indexing, that is, indexing of arrays in Java starts with 0 and not 1.

Let's take another example of the multidimensional array. This time we will be creating a 3-dimensional array. For example,

```
String[][][] data = new
String[3][4][2];
```

Here, *data* is a 3d array that can hold a maximum of 24 (3*4*2) elements of type `String`.

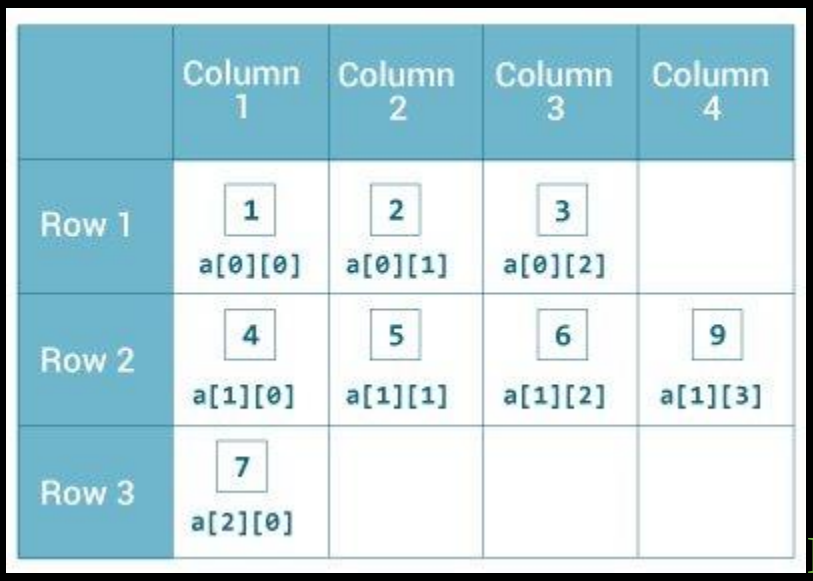
How to initialize a 2d array in Java?

Here is how we can initialize a 2-dimensional array in Java.

```
int[][] a = {
```

```
        {1, 2, 3},
        {4, 5, 6, 9},
        {7},
};
```

As we can see, each element of the multidimensional array is an array itself. And also, unlike C/C++, each row of the multidimensional array in Java can be of different lengths.



Initialization of 2-dimensional Array

Example: 2-dimensional Array

```
class MultidimensionalArray {
    public static void main(String[]
args) {

        // create a 2d array
        int[][] a = {
```

```
            {1, 2, 3},
            {4, 5, 6, 9},
            {7},
        };

        // calculate the length of
each row
        System.out.println("Length of
row 1: " + a[0].length);
        System.out.println("Length of
row 2: " + a[1].length);
        System.out.println("Length of
row 3: " + a[2].length);
    }
}
```

Run Code

Output:

```
Length of row 1: 3
Length of row 2: 4
Length of row 3: 1
```

In the above example, we are creating a multidimensional array named *a*. Since each component of a multidimensional array is also an array (`a[0]`, `a[1]` and `a[2]` are also arrays).

Here, we are using the `length` attribute to calculate the length of each row.

Example: Print all elements of 2d array Using Loop

```
class MultidimensionalArray {
    public static void main(String[]
args) {

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

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

System.out.println(a[i][j]);
            }
        }
    }
}
```

Run Code

Output:

```
1
-2
3
-4
-5
6
9
7
```

We can also use the for...each loop to access elements of the multidimensional array. For example,

```
class MultidimensionalArray {
    public static void main(String[]
args) {

        // create a 2d array
        int[][] a = {
            {1, -2, 3},
            {-4, -5, 6, 9},
            {7},
        };

        // first for...each loop
access the individual array
        // inside the 2d array
        for (int[] innerArray: a) {
```

```
            // second for...each loop
access each element inside the row
            for(int data: innerArray)
{

System.out.println(data);
            }
        }
    }
}
```

Run Code

Output:

```
1
-2
3
-4
-5
6
9
7
```

In the above example, we are have created a 2d array named *a*. We then used `for` loop and `for...each` loop to access each element of the array.

How to initialize a 3d array in Java?

Let's see how we can use a 3d array in Java. We can initialize a 3d array similar to the 2d array. For example,

```
// test is a 3d array
int[][][] test = {
          {
            {1, -2, 3},
            {2, 3, 4}
          },
          {
            {-4, -5, 6, 9},
            {1},
            {2, 3}
          }
};
```

Basically, a 3d array is an array of 2d arrays. The rows of a 3d array can also vary in length just like in a 2d array.

Example: 3-dimensional Array

```
class ThreeArray {
    public static void main(String[]
args) {

        // create a 3d array
        int[][][] test = {
            {
```

```
                {1, -2, 3},
                {2, 3, 4}
            },
            {
                {-4, -5, 6, 9},
                {1},
                {2, 3}
            }
        };

        // for..each loop to iterate
through elements of 3d array
        for (int[][] array2D: test) {
            for (int[] array1D:
array2D) {
                for(int item: array1D)
{

System.out.println(item);
                }
            }
        }
    }
}
```

Run Code

Output:

```
1
```

```
-2
3
2
3
4
-4
-5
6
9
1
2
3
```

Java Arrays

In this tutorial, we will learn to work with arrays in Java. We will learn to declare, initialize, and access array elements with the help of examples.

An array is a collection of similar types of data.

For example, if we want to store the names of 100 people then we can create an array of the string type that can store 100 names.

```
String[] array = new String[100];
```

Here, the above array cannot store more than 100 names. The number of values in a Java array is always fixed.

How to declare an array in Java?

In Java, here is how we can declare an array.

```
dataType[] arrayName;
```

- *dataType* - it can be primitive data types like `int`, `char`, `double`, `byte`, etc. or Java objects
- *arrayName* - it is an identifier

For example,

```
double[] data;
```

Here, *data* is an array that can hold values of type `double`.

But, how many elements can array this hold?

Good question! To define the number of elements that an array can hold, we have to allocate memory for the array in Java. For example,

```
// declare an array
double[] data;

// allocate memory
data = new double[10];
```

Here, the array can store **10** elements. We can also say that the **size or length** of the array is 10.

In Java, we can declare and allocate the memory of an array in one single statement. For example,

```
double[] data = new double[10];
```

How to Initialize Arrays in Java?

In Java, we can initialize arrays during declaration. For example,

```
//declare and initialize and array
int[] age = {12, 4, 5, 2, 5};
```

Here, we have created an array named age and initialized it with the values inside the curly brackets.

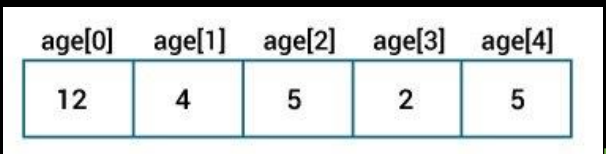
Note that we have not provided the size of the array. In this case, the Java compiler automatically specifies the size by counting the number of elements in the array (i.e. 5).

In the Java array, each memory location is associated with a number. The number is known as an array index. We can also initialize arrays in Java, using the index number. For example,

```
// declare an array
```

```
int[] age = new int[5];

// initialize array
age[0] = 12;
age[1] = 4;
age[2] = 5;
..
```



Java Arrays initialization

Note:

- Array indices always start from 0. That is, the first element of an array is at index 0.
- If the size of an array is *n*, then the last element of the array will be at index *n-1*.

How to Access Elements of an Array in Java?

We can access the element of an array using the index number. Here is the syntax for accessing elements of an array,

```
// access array elements
array[index]
```

Let's see an example of accessing array elements using index numbers.

Example: Access Array Elements

```
class Main {
 public static void main(String[]
args) {

    // create an array
    int[] age = {12, 4, 5, 2, 5};

    // access each array elements
    System.out.println("Accessing
Elements of Array:");
    System.out.println("First Element:
" + age[0]);
    System.out.println("Second Element:
" + age[1]);
    System.out.println("Third Element:
" + age[2]);
    System.out.println("Fourth Element:
" + age[3]);
    System.out.println("Fifth Element:
" + age[4]);
  }
}
```

Run Code

Output

```
Accessing Elements of Array:
```

```
First Element: 12
Second Element: 4
Third Element: 5
Fourth Element: 2
Fifth Element: 5
```

In the above example, notice that we are using the index number to access each element of the array.

We can use loops to access all the elements of the array at once.

Looping Through Array Elements

In Java, we can also loop through each element of the array. For example,

Example: Using For Loop

```
class Main {
 public static void main(String[]
args) {

    // create an array
    int[] age = {12, 4, 5};

    // loop through the array
    // using for loop
```

```
    System.out.println("Using for
Loop:");
    for(int i = 0; i < age.length; i++)
{
      System.out.println(age[i]);
    }
  }
}
```

Run Code

Output

```
Using for Loop:
12
4
5
```

In the above example, we are using the for Loop in Java to iterate through each element of the array. Notice the expression inside the loop,

```
age.length
```

Here, we are using the `length` property of the array to get the size of the array.

We can also use the for-each loop to iterate through the elements of an array. For example,

Example: Using the for-each Loop

```
class Main {
 public static void main(String[]
args) {

    // create an array
    int[] age = {12, 4, 5};

    // loop through the array
    // using for loop
    System.out.println("Using for-each
Loop:");
    for(int a : age) {
      System.out.println(a);
    }
 }
}
```

Run Code

Output

```
Using for-each Loop:
12
4
5
```

Example: Compute Sum and Average of Array Elements

```
class Main {
```

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

   int[] numbers = {2, -9, 0, 5, 12, -
25, 22, 9, 8, 12};
   int sum = 0;
   Double average;

   // access all elements using for
each loop
   // add each element in sum
   for (int number: numbers) {
     sum += number;
   }

   // get the total number of elements
   int arrayLength = numbers.length;

   // calculate the average
   // convert the average from int to
double
   average =  ((double)sum /
(double)arrayLength);

   System.out.println("Sum = " + sum);
   System.out.println("Average = " +
average);
 }
```

```
}
```

Run Code

Output:

```
Sum = 36
Average = 3.6
```

In the above example, we have created an array of named *numbers*. We have used the `for...each` loop to access each element of the array.

Inside the loop, we are calculating the sum of each element. Notice the line,

```
int arrayLength = number.length;
```

Here, we are using the length attribute of the array to calculate the size of the array. We then calculate the average using:

```
average = ((double)sum /
(double)arrayLength);
```

As you can see, we are converting the `int` value into `double`. This is called type casting in Java. To learn more about typecasting, visit Java Type Casting.

Multidimensional Arrays

Arrays we have mentioned till now are called one-dimensional arrays. However, we can declare multidimensional arrays in Java.

A multidimensional array is an array of arrays. That is, each element of a multidimensional array is an array itself. For example,

```
double[][] matrix = {{1.2, 4.3, 4.0},
      {4.1, -1.1}
};
```

Here, we have created a multidimensional array named matrix. It is a 2-dimensional array. To learn more, visit the Java multidimensional array.

Recommended Readings

- Java Copy Array
- Java Program to Print an Array
- Java Program to Concatenate two Arrays
- Java ArrayList to Array and Array to ArrayList
- Java Dynamic Array