**Chapter 3 : Java Arrays**

Normally, an array is a collection of similar type of elements which has contiguous memory location.

****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.

Unlike C/C++, we can get the length of the array using the length member. In C/C++, we need to use the sizeof operator.

In Java, array is an object of a dynamically generated class. Java array inherits the Object class, and implements the Serializable as well as Cloneable interfaces. We can store primitive values or objects in an array in Java. Like C/C++, we can also create single dimentional or multidimentional arrays in Java.

Moreover, Java provides the feature of anonymous arrays which is not available in C/C++.



### 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.

### Types of Array in java

There are two types of array.

* Single Dimensional Array
* Multidimensional Array

## 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];

### Example of Java Array

Let's see the simple example of java array, where we are going to declare, instantiate, initialize and traverse an array.

1. //Java Program to illustrate how to declare, instantiate, initialize
2. //and traverse the Java array.
3. **class** Testarray{
4. **public** **static** **void** main(String args[]){
5. **int** a[]=**new** **int**[5];//declaration and instantiation
6. a[0]=10;//initialization
7. a[1]=20;
8. a[2]=70;
9. a[3]=40;
10. a[4]=50;
11. //traversing array
12. **for**(**int** i=0;i<a.length;i++)//length is the property of array
13. System.out.println(a[i]);
14. }}

Output:

10

20

70

40

50

## Declaration, Instantiation and Initialization of Java Array

We can declare, instantiate and initialize the java array together by:

1. **int** a[]={33,3,4,5};//declaration, instantiation and initialization

Let's see the simple example to print this array.

1. //Java Program to illustrate the use of declaration, instantiation
2. //and initialization of Java array in a single line
3. **class** Testarray1{
4. **public** **static** **void** main(String args[]){
5. **int** a[]={33,3,4,5};//declaration, instantiation and initialization
6. //printing array
7. **for**(**int** i=0;i<a.length;i++)//length is the property of array
8. System.out.println(a[i]);
9. }}

Output:

33

3

4

5

## For-each Loop for Java Array

We can also print the Java array using **[for-each loop](https://www.javatpoint.com/for-each-loop)**. The Java for-each loop prints the array elements one by one. It holds an array element in a variable, then executes the body of the loop.

The syntax of the for-each loop is given below:

1. **for**(data\_type variable:array){
2. //body of the loop
3. }

Let us see the example of print the elements of Java array using the for-each loop.

1. //Java Program to print the array elements using for-each loop
2. **class** Testarray1{
3. **public** **static** **void** main(String args[]){
4. **int** arr[]={33,3,4,5};
5. //printing array using for-each loop
6. **for**(**int** i:arr)
7. System.out.println(i);
8. }}

Output:

33

3

4

5

## Passing Array to a Method in Java

We can pass the java array to method so that we can reuse the same logic on any array.

Let's see the simple example to get the minimum number of an array using a method.

1. //Java Program to demonstrate the way of passing an array
2. //to method.
3. **class** Testarray2{
4. //creating a method which receives an array as a parameter
5. **static** **void** min(**int** arr[]){
6. **int** min=arr[0];
7. **for**(**int** i=1;i<arr.length;i++)
8. **if**(min>arr[i])
9. min=arr[i];
11. System.out.println(min);
12. }
14. **public** **static** **void** main(String args[]){
15. **int** a[]={33,3,4,5};//declaring and initializing an array
16. min(a);//passing array to method
17. }}

Output:

3

## Anonymous Array in Java

Java supports the feature of an anonymous array, so you don't need to declare the array while passing an array to the method.

1. //Java Program to demonstrate the way of passing an anonymous array
2. //to method.
3. **public** **class** TestAnonymousArray{
4. //creating a method which receives an array as a parameter
5. **static** **void** printArray(**int** arr[]){
6. **for**(**int** i=0;i<arr.length;i++)
7. System.out.println(arr[i]);
8. }
10. **public** **static** **void** main(String args[]){
11. printArray(**new** **int**[]{10,22,44,66});//passing anonymous array to method
12. }}

Output:

10

22

44

66

## Returning Array from the Method

We can also return an array from the method in Java.

1. //Java Program to return an array from the method
2. **class** TestReturnArray{
3. //creating method which returns an array
4. **static** **int**[] get(){
5. **return** **new** **int**[]{10,30,50,90,60};
6. }
8. **public** **static** **void** main(String args[]){
9. //calling method which returns an array
10. **int** arr[]=get();
11. //printing the values of an array
12. **for**(**int** i=0;i<arr.length;i++)
13. System.out.println(arr[i]);
14. }}

Output:

10

30

50

90

60

## ArrayIndexOutOfBoundsException

The Java Virtual Machine (JVM) throws an ArrayIndexOutOfBoundsException if length of the array in negative, equal to the array size or greater than the array size while traversing the array.

1. //Java Program to demonstrate the case of
2. //ArrayIndexOutOfBoundsException in a Java Array.
3. **public** **class** TestArrayException{
4. **public** **static** **void** main(String args[]){
5. **int** arr[]={50,60,70,80};
6. **for**(**int** i=0;i<=arr.length;i++){
7. System.out.println(arr[i]);
8. }
9. }}

Output:

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4

at TestArrayException.main(TestArrayException.java:5)

50

60

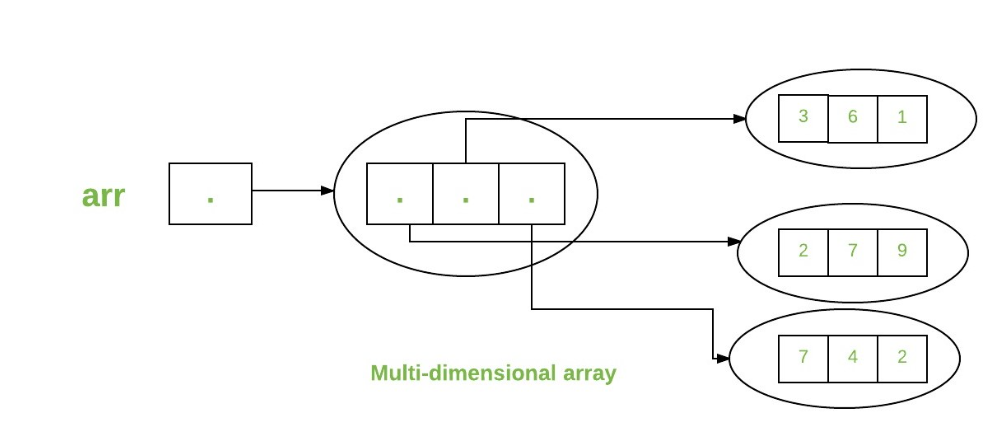
70

80

## Multidimensional Array in Java

In such case, data is stored in row and column based index (also known as matrix form).

****Syntax to Declare Multidimensional Array in Java****



### Example of Multidimensional Java Array

Let's see the simple example to declare, instantiate, initialize and print the 2Dimensional array.

1. //Java Program to illustrate the use of multidimensional array
2. **class** Testarray3{
3. **public** **static** **void** main(String args[]){
4. //declaring and initializing 2D array
5. **int** arr[][]={{1,2,3},{2,4,5},{4,4,5}};
6. //printing 2D array
7. **for**(**int** i=0;i<3;i++){
8. **for**(**int** j=0;j<3;j++){
9. System.out.print(arr[i][j]+" ");
10. }
11. System.out.println();
12. }
13. }}

Output:

1 2 3

2 4 5

4 4 5

## Jagged Array in Java

If we are creating odd number of columns in a 2D array, it is known as a jagged array. In other words, it is an array of arrays with different number of columns.

1. //Java Program to illustrate the jagged array
2. **class** TestJaggedArray{
3. **public** **static** **void** main(String[] args){
4. //declaring a 2D array with odd columns
5. **int** arr[][] = **new** **int**[3][];
6. arr[0] = **new** **int**[3];
7. arr[1] = **new** **int**[4];
8. arr[2] = **new** **int**[2];
9. //initializing a jagged array
10. **int** count = 0;
11. **for** (**int** i=0; i<arr.length; i++)
12. **for**(**int** j=0; j<arr[i].length; j++)
13. arr[i][j] = count++;
15. //printing the data of a jagged array
16. **for** (**int** i=0; i<arr.length; i++){
17. **for** (**int** j=0; j<arr[i].length; j++){
18. System.out.print(arr[i][j]+" ");
19. }
20. System.out.println();//new line
21. }
22. }
23. }

Output:

0 1 2

3 4 5 6

7 8

## What is the class name of Java array?

In Java, an array is an object. For array object, a proxy class is created whose name can be obtained by getClass().getName() method on the object.

1. //Java Program to get the class name of array in Java
2. **class** Testarray4{
3. **public** **static** **void** main(String args[]){
4. //declaration and initialization of array
5. **int** arr[]={4,4,5};
6. //getting the class name of Java array
7. Class c=arr.getClass();
8. String name=c.getName();
9. //printing the class name of Java array
10. System.out.println(name);
12. }}

Output:

I

## Copying a Java Array

We can copy an array to another by the arraycopy() method of System class.

****Syntax of arraycopy method****

1. **public** **static** **void** arraycopy(
2. Object src, **int** srcPos,Object dest, **int** destPos, **int** length
3. )

### Example of Copying an Array in Java

1. //Java Program to copy a source array into a destination array in Java
2. **class** TestArrayCopyDemo {
3. **public** **static** **void** main(String[] args) {
4. //declaring a source array
5. **char**[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e',
6. 'i', 'n', 'a', 't', 'e', 'd' };
7. //declaring a destination array
8. **char**[] copyTo = **new** **char**[7];
9. //copying array using System.arraycopy() method
10. System.arraycopy(copyFrom, 2, copyTo, 0, 7);
11. //printing the destination array
12. System.out.println(String.valueOf(copyTo));
13. }
14. }

Output:

caffein

## Cloning an Array in Java

1. //Java Program to clone the array
2. **class** Testarray1{
3. **public** **static** **void** main(String args[]){
4. **int** arr[]={33,3,4,5};
5. System.out.println("Printing original array:");
6. **for**(**int** i:arr)
7. System.out.println(i);
9. System.out.println("Printing clone of the array:");
10. **int** carr[]=arr.clone();
11. **for**(**int** i:carr)
12. System.out.println(i);

15. }}

Output:

Printing original array:

33

3

4

5

Printing clone of the array:

33

3

4

5

## 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 + ", ");

}

}

}

****Output****:

1, 2, 3, 4, 5, 6

## 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));

}

}

****Output****:

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

### Cloning of arrays

When you clone a single-dimensional array, such as Object[], a “deep copy” is performed with the new array containing copies of the original array’s elements as opposed to references.

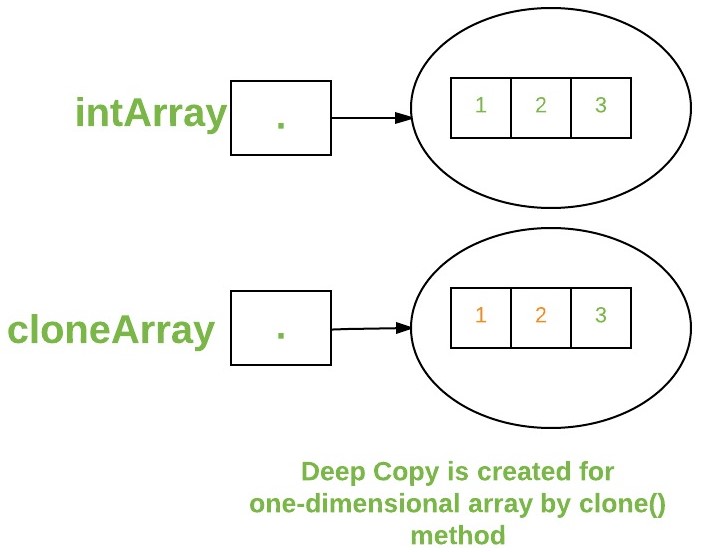
* Java

|  |
| --- |
| // Java program to demonstrate  // cloning of one-dimensional arrays    **class** Test {  **public** **static** **void** main(String args[])      {  **int** intArray[] = { 1, 2, 3 };    **int** cloneArray[] = intArray.clone();            // will print false as deep copy is created          // for one-dimensional array          System.out.println(intArray == cloneArray);    **for** (**int** i = 0; i < cloneArray.length; i++) {              System.out.print(cloneArray[i] + " ");          }      }  } |

**Output**

false

1 2 3



A clone of a multi-dimensional array (like Object[][]) is a “shallow copy,” however, which is to say that it creates only a single new array with each element array a reference to an original element array, but subarrays are shared.

* Java

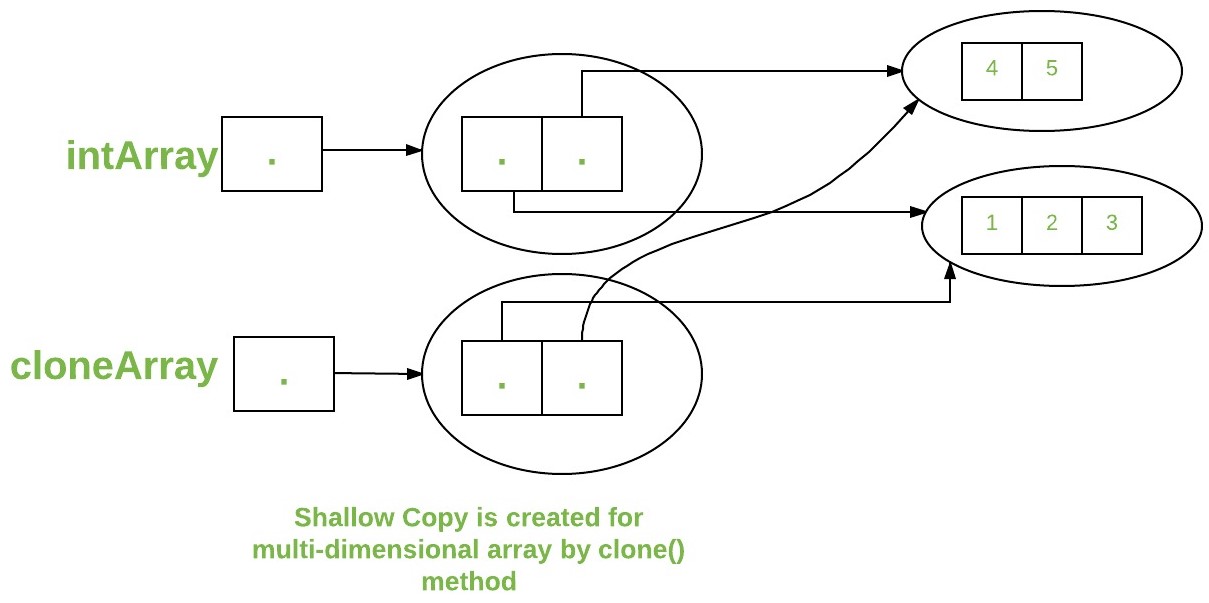
|  |
| --- |
| // Java program to demonstrate  // cloning of multi-dimensional arrays    **class** Test {  **public** **static** **void** main(String args[])      {  **int** intArray[][] = { { 1, 2, 3 }, { 4, 5 } };    **int** cloneArray[][] = intArray.clone();            // will print false          System.out.println(intArray == cloneArray);            // will print true as shallow copy is created          // i.e. sub-arrays are shared          System.out.println(intArray[0] == cloneArray[0]);          System.out.println(intArray[1] == cloneArray[1]);      }  } |

**Output**

false

true

true



### Arrays of Objects

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

Student[] arr = new Student[5]; //student is a user-defined class

**Syntax:**

1) data type[] arrName;

2) datatype arrName[];

3) datatype [] arrName;

The student Array contains five memory spaces each of the size of student class in which the address of five Student objects can be stored. The Student objects have to be instantiated using the constructor of the Student class, and their references should be assigned to the array elements in the following way.

**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 the array are objects of a class Student.

**public** **class** GFG {

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

    {

        // declares an Array of integers.

        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