# Arrays in Java

Last Updated : 03 Feb, 2025

**Arrays** are fundamental structures in Java that allow us to store multiple values of the same type in a single variable. They are useful for storing and managing collections of data. Arrays in Java are objects, which makes them work differently from arrays in C/C++ in terms of memory management.

For **primitive arrays**, **elements** are stored in a contiguous memory location. For **non-primitive arrays**, **references** are stored at contiguous locations, but the actual objects may be at different locations in memory.

**Example:**

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

        // initializing array
        int[] arr = { 1, 2, 3, 4, 5 };

        // size of array
        int n = arr.length;

        // traversing array
        for (int i = 0; i < n; i++)
            System.out.print(arr[i] + " ");
    }
}
```

**Output**

```
1 2 3 4 5
```

## Basics of Arrays in Java

There are some basic operations we can start with as mentioned below:

### 1. Array Declaration

To declare an array in Java, use the following syntax:

```
type[] arrayName;
```

- **type**: The data type of the array elements (e.g., `int`, `String`).
- **arrayName**: The name of the array.

**Note:** The array is not yet initialized.

### 2. Create an Array

To create an array, you need to allocate memory for it using the `new` keyword:

```
// Creating an array of 5 integers
int[] numbers = new int[5];
```

This statement initializes the `numbers` array to hold 5 integers. The default value for each element is `0`.

### 3. Access an Element of an Array

We can access array elements using their index, which starts from `0`:

```
// Setting the first element of the array
numbers[0] = 10;

// Accessing the first element
int firstElement = numbers[0];
```

The first line sets the value of the first element to `10`. The second line retrieves the value of the first element.

### 4. Change an Array Element

To change an element, assign a new value to a specific index:

> *// Changing the first element to 20*
> *numbers[0] = 20;*

### 5. Array Length

We can get the length of an array using the `length` property:

> *// Getting the length of the array*
> *int length = numbers.length;*

Now, we have completed with basic operations so let us go through the **in-depth concepts of Java Arrays**, through the diagrams, examples, and explanations.

## In-Depth Concepts of Java Array

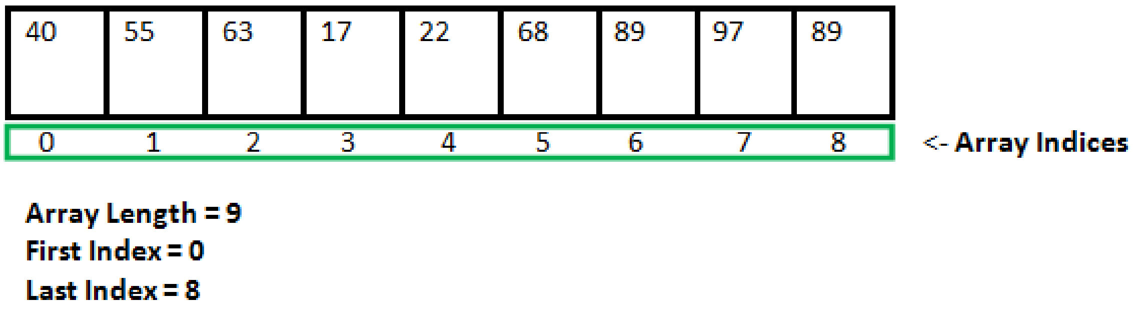
Following are some important points about Java arrays.

### Array Properties

- In Java, all arrays are dynamically allocated.
- Arrays may be stored in **contiguous memory** [consecutive memory locations].
- Since arrays are objects in Java, we can find their length using the object property *length*. This is different from C/C++, where we find length using size of.
- A Java array variable can also be declared like other variables with [] after the data type.
- The variables in the array are ordered, and each has an index beginning with 0.

- Java array can also be used as a static field, a local variable, or a method parameter.

An array can contain primitives (int, char, etc.) and object (or non-primitive) references of a class, depending on the definition of the array. In the case of primitive data types, the actual values might be stored in contiguous memory locations (JVM does not guarantee this behavior). In the case of class objects, the actual objects are stored in a heap segment.



> ***Note:*** *This storage of arrays helps us randomly access the elements of an array [Support Random Access].*

## Creating, Initializing, and Accessing an Arrays in Java

For understanding the array we need to understand how it actually works. To understand this follow the flow mentioned below:

- Declare
- Initialize
- Access

### i. Declaring an Array

The general form of array declaration is

> ***Method 1:***
> *type var-name[];*
>
> ***Method 2:***
> *type[] var-name;*

The element type determines the data type of each element that comprises the array. Like an array of integers, we can also create an array of other primitive data types like char, float, double, etc., or user-defined data types (objects of a class).

**Note:** It is just how we can create is an array variable, **no actual array exists**. It merely tells the compiler that this variable (int Array) will hold an array of the integer type.

Now, Let us provide memory storage to this created array.

## ii. Initialization an Array in Java

When an array is declared, only a reference of an array is created. The general form of *new* as it applies to one-dimensional arrays appears as follows:

```
var-name = new type [size];
```

Here, *type* specifies the type of data being allocated, *size* determines the number of elements in the array, and *var-name* is the name of the array variable that is linked to the array. To use *new* to allocate an array, **you must specify the type and number of elements to allocate.**

**Example:**

```
// declaring array
int intArray[];

// allocating memory to array
intArray = new int[20];

// combining both statements in one
int[] intArray = new int[20];
```

**Note:** The elements in the array allocated by *new* will automatically be initialized to **zero** (for numeric types), **false** (for boolean), or **null** (for reference types). Do refer to default array values in Java.

Obtaining an array is a two-step process. First, you must declare a variable of the desired array type. Second, you must allocate the memory to hold the array, using new, and assign it to the array variable. Thus, **in Java**, **all arrays are dynamically allocated.**

### Array Literal in Java

In a situation where the size of the array and variables of the array are already known, array literals can be used.

> *// Declaring array literal*
> *int[] intArray = new int[]{ 1,2,3,4,5,6,7,8,9,10 };*

- The length of this array determines the length of the created array.
- There is no need to write the new int[] part in the latest versions of Java.

### iii. Accessing Java Array Elements using for Loop

Now , we have created an Array with or without the values stored in it. Access becomes an important part to operate over the values mentioned within the array indexes using the points mentioned below:

- Each element in the array is accessed via its index.
- The index begins with 0 and ends at (total array size)-1.
- All the elements of array can be accessed using Java for Loop.

Let us check the syntax of basic for loop to traverse an array:

> *// Accessing the elements of the specified array*
> *for (int i = 0; i < arr.length; i++)*
> *System.out.println("Element at index " + i + " : "+ arr[i]);*

**Implementation:**

```
// Java program to illustrate creating an array
// of integers,  puts some values in the array,
// and prints each value to standard output.
```

```
class GFG {
    public static void main(String[] args)
    {
        // declares an Array of integers.
        int[] arr;

        // allocating memory for 5 integers.
        arr = new int[5];

        // initialize the elements of the array
        // first to last(fifth) element
          arr[0] = 10;
        arr[1] = 20;
        arr[2] = 30;
        arr[3] = 40;
        arr[4] = 50;

        // accessing the elements of the specified array
        for (int i = 0; i < arr.length; i++)
            System.out.println("Element at index "
                                   + i + " : " + arr[i]);
    }
}
```

### Output

```
Element at index 0 : 10
Element at index 1 : 20
Element at index 2 : 30
Element at index 3 : 40
Element at index 4 : 50
```

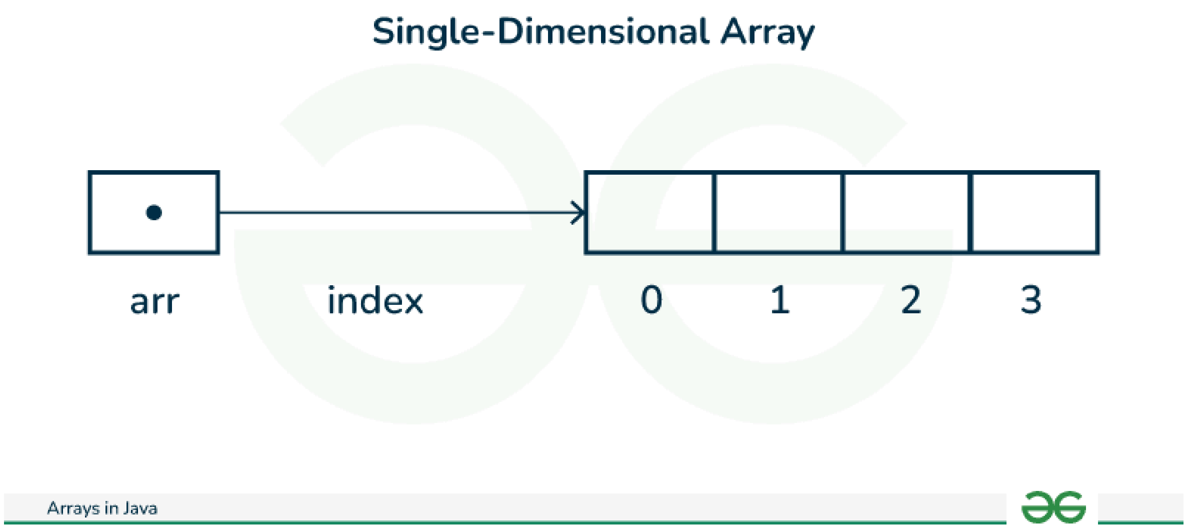
## Types of Arrays in Java

Java supports different types of arrays:

### 1. Single-Dimensional Arrays

These are the most common type of arrays, where elements are stored in a linear order.

> ***// A single-dimensional array***
> *int[] singleDimArray = {1, 2, 3, 4, 5};*



### 2. Multi-Dimensional Arrays

Arrays with more than one dimension, such as two-dimensional arrays (matrices).

```
// A 2D array (matrix)
int[][] multiDimArray = {
     {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9} };
```

You can also access java arrays using for each loops.

## Arrays of Objects in Java

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

### Syntax:

***Method 1:***
*ObjectType[] arrName;*

***Method 2:***
*ObjectType arrName[];*

## Example of Arrays of Objects

**Example 1:** Here we are taking a student class and creating an array of Student with five Student objects stored in the array. The Student objects have to be instantiated using the constructor of the Student class, and their references should be assigned to the array elements.

```java
// Java program to illustrate creating
//  an array of objects

class Student {
    public int roll_no;
    public String name;

    Student(int roll_no, String name){
        this.roll_no = roll_no;
        this.name = name;
    }
}

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

        // declares an Array of Student
        Student[] arr;

        // allocating memory for 5 objects of type Student.
        arr = new Student[5];

        // initialize the elements of the array
        arr[0] = new Student(1, "aman");
        arr[1] = new Student(2, "vaibhav");
        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 }
```

**Example 2:** An array of objects is also created like

```java
// Java program to illustrate creating
//  an array of objects

class Student{
    public String name;

    Student(String name){
        this.name = name;
    }

      @Override
    public String toString(){
        return name;
    }
}


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

        // declares an Array and initializing the
          // elements of the array
        Student[] myStudents = new Student[]{
          new Student("Dharma"),new Student("sanvi"),
          new Student("Rupa"),new Student("Ajay")
        };

        // accessing the elements of the specified array
        for(Student m:myStudents){
            System.out.println(m);
        }
    }
}
```

### Output

```
Dharma
sanvi
Rupa
Ajay
```

## What happens if we try to access elements 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.

### Below code shows what happens if we try to access elements outside the array size:

```java
// Code for showing error "ArrayIndexOutOfBoundsException"

public class GFG {
    public static void main(String[] args)
    {
        int[] arr = new int[4];
        arr[0] = 10;
        arr[1] = 20;
        arr[2] = 30;
        arr[3] = 40;

        System.out.println(
            "Trying to access element outside the size of array");
        System.out.println(arr[5]);
    }
}
```

### Output

```
Trying to access element outside the size of array
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds
for length 4
at GFG.main(GFG.java:13)
```

## Multidimensional Arrays in Java

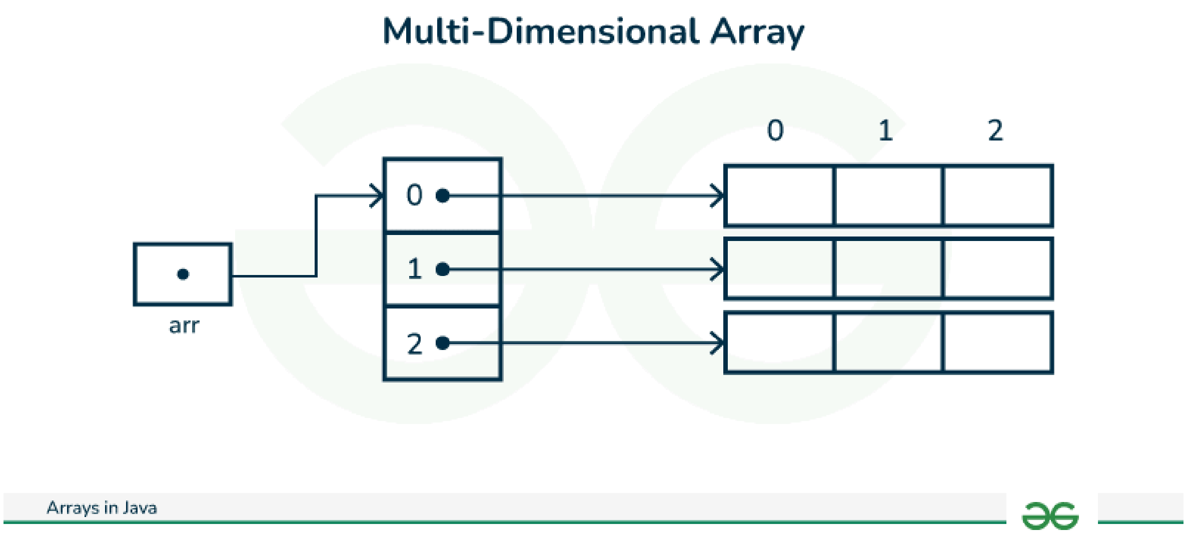
Multidimensional arrays are **arrays of arrays** with each element of the array holding the reference of other arrays. These are also known as Jagged Arrays. A multidimensional array is created by appending one set of square brackets ([]) per dimension.

### Syntax:

There are **2 methods** to declare Java Multidimensional Arrays as mentioned below:

*// Method 1*
*datatype [][] arrayrefvariable;*

*// Method 2*
*datatype arrayrefvariable[][];*



## Declaration:

***// 2D array or matrix***
*int[][] intArray = new int[10][20];*

***// 3D array***
*int[][][] intArray = new int[10][20][10];*

## Java Multidimensional Arrays Examples

**Example 1:** Let us start with basic two dimensional Array declared and initialized.

```
// Java Program to demonstrate
// Multidimensional Array
import java.io.*;

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

        // Two Dimensional Array
          // Declared and Initialized
          int[][] arr = new int[3][3];


        // Number of Rows
        System.out.println("Rows : " + arr.length);

        // Number of Columns
        System.out.println("Columns : " + arr[0].length);
    }
}
```

### Output

```
Rows:3
Columns:3
```

**Example 2:** Now, after declaring and initializing the array we will check how to Traverse the Multidimensional Array using for loop.

```java
// Java Program to Multidimensional Array

// Driver Class
public class multiDimensional {
      // main function
    public static void main(String args[])
    {
        // declaring and initializing 2D array
        int arr[][] = { { 2, 7, 9 }, { 3, 6, 1 }, { 7, 4, 2 } };

        // printing 2D array
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++)
                System.out.print(arr[i][j] + " ");

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

### Output

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

## Passing Arrays to Methods

Like variables, we can also pass arrays to methods. For example, the below program passes the array to method *sum* to calculate the sum of the array's values.

```java
// Java program to demonstrate
// passing of array to method

public class Test {
    // Driver method
    public static void main(String args[])
    {
        int arr[] = { 3, 1, 2, 5, 4 };
        // passing array to method m1
        sum(arr);
    }

    public static void sum(int[] arr)
    {
        // getting sum of array values
        int sum = 0;

        for (int i = 0; i < arr.length; i++)
            sum += arr[i];

        System.out.println("sum of array values : " + sum);
    }
}
```

### Output

```
sum of array values : 15
```

## Returning Arrays from Methods

As usual, a method can also return an array. For example, the below program returns an array from method *m1*.

```
// Java program to demonstrate
// return of array from method

class Test {
    // Driver method
    public static void main(String args[])
    {
        int arr[] = m1();

        for (int i = 0; i < arr.length; i++)
            System.out.print(arr[i] + " ");
    }

    public static int[] m1()
    {
        // returning  array
        return new int[] { 1, 2, 3 };
    }
}
```

### Output

```
1 2 3
```

## Java Array Members

Now, as you know that arrays are objects of a class, and a direct superclass of arrays is a class Object.

The members of an array type are all of the following:

- The public final field *length* contains the number of components of the array. Length may be positive or zero.
- All the members are inherited from class Object; the only method of Object that is not inherited is its clone method.
- The public method *clone()* overrides the clone method in class Object and throws no checked exceptions.

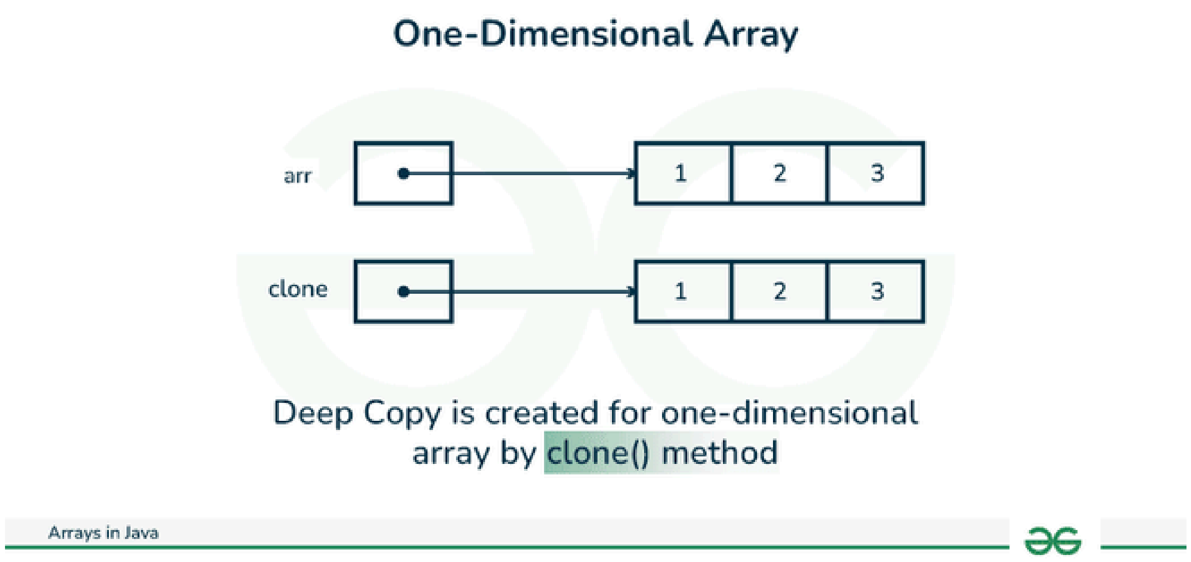
### Arrays Types and Their Allowed Element Types

| Array Types | Allowed Element Types |
|---|---|
| Primitive Type Arrays | Any type which can be implicitly promoted to declared type. |
| Object Type Arrays | Either declared type objects or it's child class objects. |
| Abstract Class Type Arrays | Its child-class objects are allowed. |
| Interface Type Arrays | Its implementation class objects are allowed. |

## Cloning Arrays in Java

### 1. Cloning of Single-Dimensional Array

When you clone a single-dimensional array, such as `Object[]`, a **shallow copy** is performed. This means that the new array contains references to the original array's elements rather than copies of the objects themselves. A deep copy occurs only with arrays containing primitive data types, where the actual values are copied.


One-Dimensional Array
arr
1
2
3
clone
1
2
3
Deep Copy is created for one-dimensional array by clone() method
Arrays in Java

**Below is the implementation of the above method:**

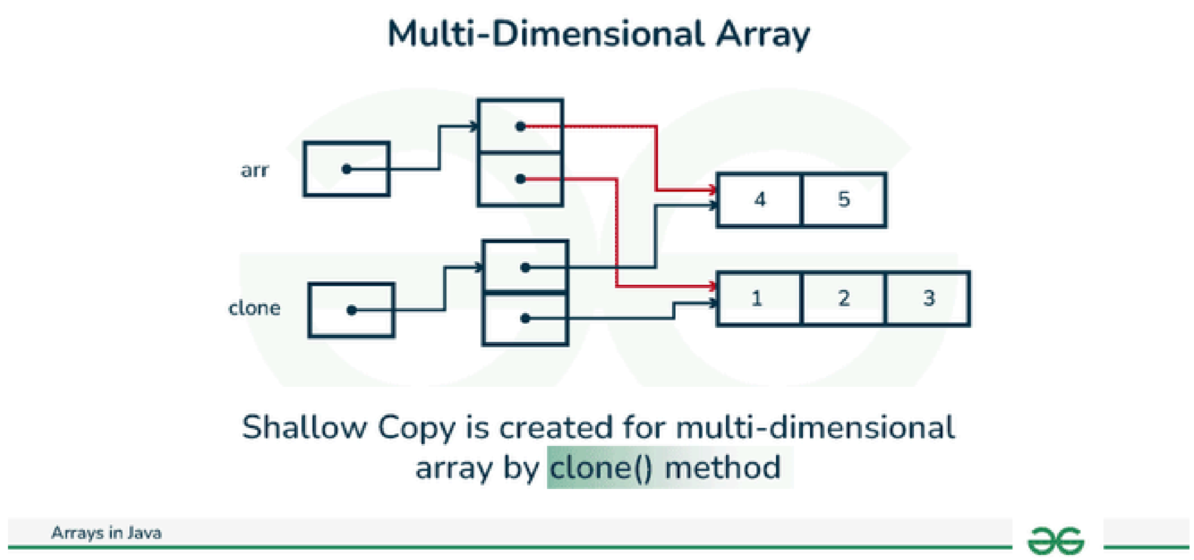
```java
// Java program to demonstrate
// cloning of one-dimensional arrays

class Test {
    public static void main(String args[])
    {
        int intArray[] = { 1, 2, 3 };

        int cloneArray[] = intArray.clone();

        // will print false as shallow copy is created
        System.out.println(intArray == cloneArray);

        for (int i = 0; i < cloneArray.length; i++) {
            System.out.print(cloneArray[i] + " ");
        }
    }
}
```

### Output

```
false
1 2 3
```

## 2. Cloning Multidimensional Array

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.


Multi-Dimensional Array
arr
clone
4
5
1
2
3
Shallow Copy is created for multi-dimensional array by clone() method
Arrays in Java

### Below is the implementation of the above method:

```
// 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
```

## Advantages of Java Arrays

- **Efficient Access**: Accessing an element by its index is fast and has constant time complexity, O(1).
- **Memory Management**: Arrays have fixed size, which makes memory management straightforward and predictable.

- **Data Organization**: Arrays help organize data in a structured manner, making it easier to manage related elements.

## Disadvantages of Java Arrays

- **Fixed Size**: Once an array is created, its size cannot be changed, which can lead to memory waste if the size is overestimated or insufficient storage if underestimated.
- **Type Homogeneity**: Arrays can only store elements of the same data type, which may require additional handling for mixed types of data.
- **Insertion and Deletion**: Inserting or deleting elements, especially in the middle of an array, can be costly as it may require shifting elements.

### Must Read:

- *Jagged Array in Java*
- *For-each loop in Java*
- *Arrays class in Java*

## Frequently Asked Questions – Java Arrays

### How can we initialize an array in Java?

*Arrays in Java can be initialized in several ways:*

- ***Static Initialization**:* `int[] arr = {1, 2, 3};`
- ***Dynamic Initialization**:* `int[] arr = new int[5];`
- ***Initialization with a loop**:* `for (int i = 0; i < arr.length; i++) { arr[i] = i + 1; }`

### Can we use an array of primitive types in Java?

*Yes, Java supports arrays of primitive types such as* `int`*,* `char`*,* `boolean`*, etc., as well as arrays of objects.*

## How are multidimensional arrays represented in Java?

*Multidimensional arrays in Java are represented as arrays of arrays. For example, a two-dimensional array is declared as `int[][] array`, and it is effectively an array where each element is another array.*

## Can we change the size of an array after it is created in Java?

*No, the size of an array in Java cannot be changed once it is initialized. Arrays are fixed-size. To work with a dynamically sized collection, consider using classes from the `java.util` package, such as `ArrayList`.*

## Can we specify the size of an array as `long` in Java?

*No, we cannot specify the size of an array as `long`. The size of an array must be specified as an `int`. If a larger size is required, it must be handled using collections or other data structures.*

## What is the direct superclass of an array in Java?

*The direct superclass of an array in Java is Object . Arrays inherit methods from the `Object` class, including `toString()`, `equals()`, and `hashCode()`.*

## Which interfaces are implemented by arrays in Java?

*All arrays in Java implement two interfaces:*

- **`Cloneable`**: *Allows the array to be cloned using the `clone()` method.*