Arrays in Java

An array in Java is a **container object** that holds a fixed number of values of the same data type. It is used to store multiple values in a single variable instead of declaring separate variables for each value.

**Key Features of Arrays in Java**

* Fixed in size (cannot change after creation)
* Stores elements of the same type (e.g., all int or all String)
* Index-based (starting from 0)
* Can be single-dimensional or multi-dimensional (like 2D arrays)

**Declaration of Arrays**

// Declaration

int[] numbers; // preferred

int numbers[]; // also valid

// Initialization

numbers = new int[10]; // creates an array of size 10 (default values: 0)

**Declaration + Initialization Together**

int[] numbers = new int[10]; // with default values (0s)

int[] marks = {90, 80, 70, 60}; // directly assigning values

**Accessing Elements**

System.out.println(marks[0]); // prints 90

marks[2] = 75; // updates index 2 (70 -> 75)

**Looping Through Arrays**

**Using for loop:**

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

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

}

**Using for-each loop:**

for (int mark : marks) {

System.out.println(mark);

}

**Multi-Dimensional Arrays (2D Array Example)**

int[][] matrix = {

{1, 2, 3},

{4, 5, 6}

};

System.out.println(matrix[0][1]); // prints 2

**Common Array Operations**

| **Operation** | **Example** |
| --- | --- |
| Length of array | arr.length |
| Sorting (using Arrays) | Arrays.sort(arr); (from java.util) |
| Copying arrays | Arrays.copyOf(arr, newLength) |
| Searching | Arrays.binarySearch(arr, key) |

**Arrays class**

The Arrays class in Java is a utility class provided in the “java.util” package. It contains a collection of static methods for manipulating arrays (such as sorting, searching, comparing, and filling). It works with both primitive and object arrays.

**Importing the Arrays Class**

import java.util.Arrays;

Commonly Used Methods

**1. Sorting**

int[] arr = {5, 2, 9, 1};

Arrays.sort(arr); // arr is now {1, 2, 5, 9}

**2. Binary Search**

The array must be sorted before using binarySearch.

int[] arr = {1, 2, 5, 9};

int index = Arrays.binarySearch(arr, 5); // returns index: 2

**3. Filling**

int[] arr = new int[5];

Arrays.fill(arr, 6); // arr becomes {6,6,6,6,6}

**4. Comparing Arrays**

int[] a = {1, 2, 3};

int[] b = {1, 2, 3};

boolean result = Arrays.equals(a, b); // true

**5. Converting to String**

int[] arr = {1, 2, 3};

System.out.println(Arrays.toString(arr)); // prints [1, 2, 3]

**6. Copying Arrays**

int[] original = {1, 2, 3};

int[] copy = Arrays.copyOf(original, 5); // {1, 2, 3, 0, 0}

**7. Parallel Sort (Java 8+)**

int[] arr = {5, 2, 1, 9};

Arrays.parallelSort(arr); // like sorting, but uses multithreading

**8. Deep Equality for Multidimensional Arrays (Comparing arrays)**

int[][] a = {{1, 2}, {3, 4}};

int[][] b = {{1, 2}, {3, 4}};

System.out.println(Arrays.deepEquals(a, b)); // true

**Example Program:**

import java.util.Arrays;

public class ArrayExample {

public static void main(String[] args) {

int[] nums = {5, 3, 8, 1};

Arrays.sort(nums);

for (int n : nums) {

System.out.print(n + " ");

}

}

}

**Output:** 1 3 5 8

**Copy Array Example**

import java.util.Arrays;

public class CopyArrayExample {

public static void main(String[] args) {

int[] original = {10, 20, 30, 40, 50};

// Copy first 3 elements

int[] copy = Arrays.copyOf(original, 3);

// Print copied array

System.out.println("Copied Array: " + Arrays.toString(copy));

}

}

**Output:**

Copied Array: [10, 20, 30]

**Example: Copy Full Array**

import java.util.Arrays;

public class CopyFullArray {

public static void main(String[] args) {

int[] original = {10, 20, 30, 40, 50};

// Copy the entire array

int[] copy = Arrays.copyOf(original, original.length);

System.out.println("Original Array: " + Arrays.toString(original));

System.out.println("Copied Array: " + Arrays.toString(copy));

}

}