Java Arrays

1. What is an Array?

An **array** in Java is a data structure that stores multiple values of the **same data type** in a contiguous memory location. Arrays allow efficient access and manipulation of elements using an **index**.

Syntax to Declare an Array

```
dataType[] arrayName;  // Preferred
dataType arrayName[];  // Also valid
```

Example

```
int[] numbers;   // Declaration
numbers = new int[5];  // Memory allocation for 5 elements
```

---

2. Types of Arrays in Java

(A) One-Dimensional Array (1D)

A **1D array** stores a sequence of elements of the same type.

Example:

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

(B) Multi-Dimensional Arrays (2D, 3D, etc.)

A **multi-dimensional array** is an array of arrays.

Example (2D Array):

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

Accessing an element:

```
System.out.println(matrix[1][2]); // Output: 6
```

---

3. Array Operations

(A) Declaring, Initializing, and Accessing an Array

```
int[] arr = {5, 10, 15, 20};
System.out.println(arr[2]); // Output: 15
```

(B) Array Length

The `.length` property gives the total number of elements in an array.

```
System.out.println(arr.length); // Output: 4
```

(C) Modifying an Array

```
arr[1] = 50; // Change the second element
System.out.println(arr[1]); // Output: 50
```

---

4. Array Functions in Java

Java provides several built-in functions for arrays using the `Arrays` class from `java.util`.

(A) Printing an Array (`Arrays.toString()`)

```
import java.util.Arrays;

int[] numbers = {5, 2, 9, 1};
System.out.println(Arrays.toString(numbers));
```

Output: `[5, 2, 9, 1]`

(B) Sorting an Array (`Arrays.sort()`)

```
Arrays.sort(numbers);
System.out.println(Arrays.toString(numbers));
```

Output: `[1, 2, 5, 9]` (Sorted in ascending order)

(C) Searching an Element (`Arrays.binarySearch()`)

Binary search works only on **sorted** arrays.

```
int index = Arrays.binarySearch(numbers, 5);
System.out.println("Index of 5: " + index);
```

Output: `Index of 5: 2`

(D) Copying an Array (`Arrays.copyOf()`)

```
int[] copiedArray = Arrays.copyOf(numbers, numbers.length);
System.out.println(Arrays.toString(copiedArray));
```

Output: `[1, 2, 5, 9]`

(E) Filling an Array (`Arrays.fill()`)

```
Arrays.fill(numbers, 7);
System.out.println(Arrays.toString(numbers));
```

Output: `[7, 7, 7, 7]` (All elements are set to 7)

(F) Iterating Over an Array

1. Using a for Loop

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

2. Using an Enhanced for Loop (For-Each)

```
for (int num : numbers) {
    System.out.print(num + " ");
}
```

---

5. Multi-Dimensional Arrays

(A) Declaring and Initializing a 2D Array

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

(B) Accessing a 2D Array Element

```
System.out.println(matrix[1][2]); // Output: 6
```

(C) Iterating Over a 2D Array

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

6. Array of Objects

Arrays can store objects as well.

Example:

```
class Student {
    String name;
    int age;

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

public class Main {
    public static void main(String[] args) {
        Student[] students = new Student[2];
        students[0] = new Student("Alice", 20);
        students[1] = new Student("Bob", 22);

        for (Student s : students) {
            System.out.println(s.name + " - " + s.age);
        }
    }
}
```

Output:

```
Alice - 20
Bob - 22
```

---

7. Jagged Arrays (Irregular 2D Arrays)

A **jagged array** is a multi-dimensional array where each row can have a different number of columns.

Example:

```
int[][] jagged = new int[3][];
jagged[0] = new int[]{1, 2};
jagged[1] = new int[]{3, 4, 5};
jagged[2] = new int[]{6};

for (int[] row : jagged) {
    System.out.println(Arrays.toString(row));
}
```

Output:

```
[1, 2]
```

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

---

8. Array Limitations

- Fixed size (cannot be dynamically resized)
- No built-in methods for adding/removing elements (use `ArrayList` for dynamic resizing)

---

9. Array vs. ArrayList

Feature	Array	ArrayList
Size	Fixed	Dynamic
Performance	Faster	Slightly slower
Methods	Limited	Rich built-in methods
Data Type	Can store primitives	Stores only objects

Use **ArrayList** when you need dynamic resizing.

- Arrays store multiple elements of the same type in a contiguous memory location.
- Java provides useful functions like `Arrays.sort()`, `Arrays.copyOf()`, and `Arrays.binarySearch()`.
- Multi-dimensional and jagged arrays help represent tabular data.
- **For dynamic data structures, prefer `ArrayList` over arrays.**