Tabel of Content

1 Array in Java

An **array** is a **collection of elements** of the **same data type** stored in **contiguous memory locations**. It is used to store multiple values under a single variable name.

Think of an array as a row of lockers (indexed), where each locker holds a value of the same type.

Syntax

1. Declaration:

```
int[] numbers;       // Recommended
// OR
int numbers[];       // Also valid
```

2. Instantiation:

```
numbers = new int[5];  // Array of 5 integers (default values: 0)
```

3. Initialization:

```
numbers[0] = 10;
numbers[1] = 20;
```

4. Combined:

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

OR

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

Real-world Analogy

Imagine you run a delivery service and assign lockers for packages. Each locker is numbered (indexed). You can access a specific locker (array index) to retrieve the package (value).

Example:

```
public class ArrayDemo {
    public static void main(String[] args) {
        String[] products = {"Laptop", "Tablet", "Mobile"};

        for (int i = 0; i < products.length; i++) {
            System.out.println(products[i]);
```

```
        }
    }
}
```

Output:

Laptop
Tablet
Mobile

Types of Arrays in Java

TYPE	DESCRIPTION	EXAMPLE
SINGLE-DIMENSIONAL	Linear list of elements	int[] marks = new int[5];
MULTI-DIMENSIONAL	Array of arrays (matrix-style)	int[][] matrix = new int[3][3];
JAGGED ARRAY	Array of arrays with different lengths	int[][] arr = new int[3][];

Array Properties

- Fixed size (declared at the time of creation)
- Index starts from 0
- Can store **primitive** or **reference types**
- Default values:
  - 0 for int
  - false for boolean
  - null for objects

Common Use Cases

- Storing student marks
- Holding a list of products
- Representing 2D data (like a matrix or a chessboard)

Advantages

- Easy to use
- Memory-efficient for fixed-size data
- Fast data access using index

Limitations

- Fixed size – can't grow dynamically (use ArrayList instead for dynamic needs)
- All elements must be of the same type
- Insertion/deletion in the middle is expensive (shifting needed)

Best Practices

- Always check array.length before iterating to avoid ArrayIndexOutOfBoundsException
- For unknown sizes, use **collections** like ArrayList
- Use **enhanced for loop** for readability:

```
for (String item : products) {
    System.out.println(item);
}
```

Summary

FEATURE	DESCRIPTION
DEFINITION	Fixed-size container for same-type elements
ACCESS	Via index (starting at 0)
TYPE	Single or multi-dimensional
BETTER ALTERNATIVE (DYNAMIC)	Use ArrayList or other collections

2 Array Input/Output in Java

- **Array Input**: Taking values from the user and storing them in an array.
- **Array Output**: Displaying the values stored in the array.

Both input and output can be performed using loops like for or for-each.

Real-World Analogy

Think of a row of exam paper slots for students:

- Input: Each student drops their paper into a numbered slot (array index).
- Output: The examiner reads papers from the slots one by one.

Example: Taking Input and Printing Output of an Integer Array

```
import java.util.Scanner;

public class ArrayIOExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter size of the array: ");
```

```
        int size = scanner.nextInt();

        int[] numbers = new int[size];

        // Input
        System.out.println("Enter " + size + " numbers:");
        for (int i = 0; i < size; i++) {
            numbers[i] = scanner.nextInt();
        }

        // Output
        System.out.println("You entered:");
        for (int i = 0; i < size; i++) {
            System.out.println("Element at index " + i + ": " + numbers[i]);
        }

        scanner.close();
    }
}
```

Sample Output

Enter size of the array: 4
Enter 4 numbers:
10
20
30
40
You entered:
Element at index 0: 10
Element at index 1: 20
Element at index 2: 30
Element at index 3: 40

Using Enhanced For Loop for Output

```
for (int num : numbers) {
    System.out.println(num);
}
```

For String Array Input/Output Example

```
String[] names = new String[3];
Scanner sc = new Scanner(System.in);

System.out.println("Enter 3 names:");
for (int i = 0; i < names.length; i++) {
    names[i] = sc.nextLine();
}

System.out.println("Names entered:");
```

```
for (String name : names) {
    System.out.println(name);
}
```

Common Mistakes to Avoid

MISTAKE	FIX
arrayindexoutofboundsexception	Always use array.length for loops
forgetting to close scanner	Use scanner.close() at the end
mixing nextint() and nextline()	Use scanner.nextLine() after nextInt() to consume leftover \n

Summary Table

OPERATION	CODE SNIPPET
DECLARE ARRAY	int[] arr = new int[5];
INPUT VALUES	arr[i] = sc.nextInt(); in loop
OUTPUT VALUES	System.out.println(arr[i]); or for-each
DYNAMIC SIZE	int size = sc.nextInt(); then create array

3 2D Array in Java

A **2D array** is an array of arrays. It stores data in **rows and columns**, like a **matrix or table**.

Definition: A 2D array in Java is declared as: dataType[][] arrayName;

Real-World Analogy

Think of a **spreadsheet** or **chessboard**:

- Rows and columns hold values.
- Each cell is accessed by its row and column number (like [i][j]).

Syntax of 2D Arrays

Declaration:

```
int[][] matrix;         // Recommended
int matrix[][];         // Also valid
```

Instantiation:

```
matrix = new int[3][4];  // 3 rows and 4 columns
```

Initialization:

```
int[][] matrix = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};
```

Accessing Elements

```
matrix[0][1];  // Access element at 1st row, 2nd column (value: 2)
```

Taking Input and Printing a 2D Array

```
import java.util.Scanner;

public class TwoDArrayIO {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int[][] matrix = new int[2][3];  // 2 rows, 3 columns

        // Input
        System.out.println("Enter elements (2 rows, 3 columns):");
        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 3; j++) {
                matrix[i][j] = sc.nextInt();
            }
        }

        // Output
        System.out.println("Matrix:");
        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print(matrix[i][j] + " ");
            }
            System.out.println();
        }

        sc.close();
    }
}
```

Sample Output:

Enter elements (2 rows, 3 columns):
1 2 3
4 5 6
Matrix:
1 2 3

4 5 6

Enhanced For Loop for Output

```
for (int[] row : matrix) {
    for (int val : row) {
        System.out.print(val + " ");
    }
    System.out.println();
}
```

Common Use Cases of 2D Arrays

USE CASE	EXAMPLE
MATRIX OPERATIONS	Addition, multiplication
TABLES OR GRIDS	Marks of students (rows: students, columns: subjects)
BOARD GAMES	Chess, Sudoku, Tic Tac Toe
IMAGE REPRESENTATION	Pixel values in grayscale

Important Points

- Indexing starts at 0 → [0][0] is first row, first column.
- Default values:
  - 0 for int/float
  - false for boolean
  - null for objects
- matrix.length gives **number of rows**
  matrix[0].length gives **number of columns**

Summary

FEATURE	DESCRIPTION
DECLARATION	int[][] arr = new int[3][4];
INPUT	Nested for loop
OUTPUT	Nested for or for-each loop
USE CASES	Tables, matrices, grids, boards

4 Array Topics for Product-Based Companies

1. Core Array Handling (Basics to Intermediate)

Subtopic	Concepts & Java APIs
Array declaration & traversal	int[], Arrays.toString(), for-each loop
Array input/output	Scanner, BufferedReader, String.split()
Array update, frequency count	for loops, HashMap<Integer, Integer>
Sorting basics	Arrays.sort(), custom comparator using Comparator
Arrays utility methods	Arrays.fill(), Arrays.copyOf(), Arrays.equals()

2. Searching & Sorting Patterns

Subtopic	Techniques
Binary Search	Standard and variations (first/last occurrence, search in rotated array)
Sorting + Two-pointer	Sort first, then apply two-pointer logic (pair/triplet problems)
Lower/Upper Bound	Custom binary search implementations
Kth Smallest/Largest	PriorityQueue, QuickSelect

Java Tip: Use Arrays.binarySearch() for simple cases

3. Sliding Window & Prefix Sum

Subtopic	Techniques & Java APIs
Max sum subarray (fixed size)	Sliding window template
Longest subarray with condition	Variable window, HashMap/Set
Prefix sum array	Precompute sum, then query ranges
Range sum query	int[] prefixSum, difference of prefix

Key Java Tip: Initialize prefix sum with one extra element: int[] prefix = new int[n+1]

4. Hashing + Two Pointer Tricks

Subtopic	Techniques
Subarray with given sum	HashMap<Integer, Integer> for prefix sum count
Count of pairs with given sum	HashMap or sort + two pointer
Longest subarray with 0 sum	Prefix sum + HashMap
Two sum, Three sum, Four sum	HashMap or sorting + two/three/four pointer

5. Advanced Array Problems

Subtopic	Techniques
Kadane's Algorithm	Maximum subarray sum
Trapping Rain Water	Two-pointer or stack
Rotate array	Reversal algorithm
Merge intervals	Sorting + merging
Maximum Product Subarray	Track min and max at each step
Container With Most Water	Two pointer approach

6. Matrix (2D Array Problems)

Subtopic	Techniques
Search in 2D matrix	Binary search or row-column approach
Spiral traversal	Direction vector handling
Rotate 2D matrix	Transpose + reverse rows
Set matrix zeroes	Use 1st row/col as marker (in-place)
Number of islands	DFS/BFS on 2D grid

Sample LeetCode Problems

Problem Type	LeetCode Problem #
Subarray sum equals K	LeetCode 560
Maximum subarray	LeetCode 53 (Kadane's)
Container With Most Water	LeetCode 11
Trapping Rain Water	LeetCode 42
Rotate Image	LeetCode 48
Merge Intervals	LeetCode 56
Search in 2D Matrix	LeetCode 74
Two Sum	LeetCode 1
3Sum	LeetCode 15

Concepts to Emphasize in Java

- Use of `Arrays.sort()` with custom `Comparator`
- Efficient input handling with `BufferedReader` and `String.split()`
- HashMap, HashSet for frequency/count based problems
- Prefix Sum, Sliding Window, and Kadane's as templates

- PriorityQueue and Java 8 features (Comparator, Streams for sorting/filtering)