**What Is an Array?**

An **array** is a container that holds multiple values of the **same data type** in a single variable, instead of declaring multiple variables separately.

Think of it like a **row of mailboxes** where each box stores one item (number, string, etc.). The mailboxes are **numbered** (indexed), and you can access each one using its number (index).

 **Same Type:** All elements in an array must have the same data type (e.g., all integers, all doubles, etc.).

 **Fixed Size:** Once an array is created, its size (i.e., the number of elements it can hold) is fixed and cannot be changed.

 **Indexing:** Elements in an array are accessed by an index. In most languages including Java, array indices start at 0. So, if you have an array of size 5, the valid indices are from 0 to 4

# **Why Use an Array?**

* **Organization:** Arrays help organize data so you can easily store and retrieve items.
* **Efficiency:** Because data is stored in contiguous memory, it is efficient to access elements using their index.
* **Iteration:** Arrays work well with loops, allowing you to perform operations on each element with minimal code.

# **Why do** we **need arrays?**

Without arrays:

int mark1 = 85;

int mark2 = 90;

int mark3 = 78;

int mark4 = 92;

With arrays:

int[] marks = {85, 90, 78, 92};

Much cleaner and manageable, especially when dealing with large data sets like 100 students, temperatures, etc.

## Syntax: How to Declare Arrays in Java

Java gives you **multiple ways** to declare and initialize arrays. This approach tells the compiler that a variable is intended to hold an array, but it doesn’t create the array or allocate memory for it yet.

int[] numbers; // Declaration of an integer array

* **Explanation:**  
  The variable numbers is declared to be an array of integers. However, at this point, it doesn’t reference any actual array in memory.

## Declaration Only (no size, no values yet)

int[] arr; // Recommended style

// or

int arr[]; // Also valid (C-style)

Just tells Java: “I will create an array of integers later.”

## ****Declaration with Memory Allocation****

Here you declare the array and allocate memory for a specified number of elements.

numbers = new int[5]; // Allocates memory for 5 integers

* **Explanation:**
  + The new keyword creates a new array in memory with 5 slots (indices 0 to 4).
  + Each slot in the array is automatically initialized to a default value (for example, 0 for integers).

You can combine declaration and allocation in one line:

int[] numbers = new int[5];

## ****Declaration + Memory Allocation****

int[] arr = new int[5];

* This creates an array of size 5 (indexes 0 to 4).
* All values are initially **0** by default (for int).

You can later set values like arr[0] = 10;

## ****Declaration with Initialization (Short-hand Form)****

This method allows you to both declare and initialize the array with specific values in one step using **curly braces** {}.

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

* **Explanation:**
  + The array is declared to be of type int and is immediately initialized with the five values listed.
  + The size of the array is automatically determined by the number of elements provided.
  + **Note:** You must use curly braces {} here. If you try using square brackets [] like int[] numbers = [10, 20, 30, 40, 50];, it will result in a syntax error because square brackets are reserved for declaring the array type and for indexing, not initializing.

## ****Declaration with Explicit Initialization Using**** new

Sometimes you may want to use the new keyword along with explicit initialization of values. This format is useful when you want to emphasize the creation of a new array instance.

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

* **Explanation:**
  + Here, you’re explicitly creating a new array using the new int[] syntax and then initializing it with the values.
  + This way, the declaration is more explicit, and it can also be handy if the array is passed as an argument to a method later.

## ****Multi-dimensional Arrays****

Arrays can have more than one dimension. For example, a 2D array is an array of arrays, often used to represent matrices.

## ****Declaring a 2D Array****

int[][] matrix;

* **Explanation:**  
  The above line declares matrix as a 2D array of integers. However, no memory is allocated yet.

## ****Allocating a 2D Array****

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

* **Explanation:**  
  Memory is allocated for 3 rows, each of which is an array of 4 integers.

## ****Declaration and Initialization in One Step****

int[][] matrix = {

{1, 2, 3, 4},

{5, 6, 7, 8},

{9, 10, 11, 12}

};

* **Explanation:**  
  Here, matrix is declared and initialized with values in a nested format using curly braces for each row.

| **Style** | **Code** | **Use Case** |
| --- | --- | --- |
| Declaration only | int[] arr; | Later initialize it |
| Declaration + size | arr = new int[5]; | Create empty array of size 5 |
| Declaration + values | int[] arr = {1,2,3}; | Initialize with known values |
| Using new with values | int[] arr = new int[]{1,2,3}; | When array must be declared first |

String[] names = {"Ram", "Sita", "Lakshman"};

double[] prices = {9.99, 5.75, 2.50};

char[] vowels = {'a', 'e', 'i', 'o', 'u'};

 **Arrays** are used to store multiple elements of the **same type** in a contiguous memory location.

 **Square brackets []** are used in declarations, memory allocation, and indexing.

 **Curly braces {}** are used to initialize an array with values.

 You can declare an array by itself, allocate memory separately, or do both in one line.

 Multi-dimensional arrays (like 2D arrays) extend the concept of arrays to more complex data structures like matrices.

## 🧠 ****Accessing Array Elements (Indexing) in Java****

## ✅ ****1. Understanding Array Indexing (Simple Explanation)****

An **index** is like the address of a house in a street. Every element in the array has a fixed position (index) starting from **0**, not 1.

So in Java:

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

| **Element** | **10** | **20** | **30** | **40** | **50** |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |

* arr[0] → 10 (first element)
* arr[4] → 50 (last element)
* arr[2] → 30 (middle element)

## ✅ ****2. Java Uses Zero-Based Indexing****

That means:

* The **first element** is at index 0
* The **last element** is at index arr.length - 1

System.out.println(arr[0]); // First element

System.out.println(arr[arr.length - 1]); // Last element

## ✅ ****3. Accessing vs. Modifying Array Elements****

#### 👉 **Accessing**:

int x = arr[1]; // Gets value at index 1

System.out.println(x); // Prints 20

#### **👉 Modifying:**

arr[2] = 100; // Replaces 30 with 100 at index 2

System.out.println(arr[2]); // Prints 100

## ✅ ****4. What Happens if You Go Out of Bounds?****

If you try to access an index **that doesn’t exist**, Java will throw an **exception**.

#### Example:

System.out.println(arr[5]); // ❌ Error! Index 5 doesn’t exist

✅ Output:

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5

🧠 **Why?**  
Because valid indices are from 0 to arr.length - 1.  
So for size 5, only indices 0 to 4 are allowed.

## ****Task 1: Print first and last element of any array****

public class Task1 {

public static void main(String[] args) {

int[] arr = {5, 10, 15, 20, 25};

System.out.println("First Element: " + arr[0]);

System.out.println("Last Element: " + arr[arr.length - 1]);

}

}

📌 **Output:**

First Element: 5

Last Element: 25

🧠 **Explanation:**

* arr[0] gives the first element
* arr[arr.length - 1] safely gives the last

## ✅ ****Task 2: Replace the 3rd element with a new value****

public class Task2 {

public static void main(String[] args) {

int[] arr = {8, 16, 24, 32, 40};

arr[2] = 99; // Replacing 3rd element (index 2)

System.out.println("Updated Array:");

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

System.out.print(arr[i] + " ");

}

}

}

📌 **Output:**

Updated Array:

8 16 99 32 40

🧠 **Note:** Index 2 = 3rd position

## ✅ ****Task 3: Print all elements in reverse using indices****

public class Task3 {

public static void main(String[] args) {

int[] arr = {11, 22, 33, 44, 55};

System.out.println("Array in Reverse:");

for (int i = arr.length - 1; i >= 0; i--) {

System.out.print(arr[i] + " ");

}

}

}

📌 **Output:**

Array in Reverse:

55 44 33 22 11

🧠 **Key Trick:** Start from arr.length - 1 and go down to 0.

## ✅ ****Task 4: Try accessing**** arr[-1] ****and explain the result****

public class Task4 {

public static void main(String[] args) {

int[] arr = {100, 200, 300};

System.out.println(arr[-1]); // ❌ This line causes an error

}

}

📌 **Output/Error:**

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: -1

🧠 **Explanation:**

* Indexes in Java **must be 0 or positive integers**
* -1 is **invalid**, so Java throws an exception to protect the program