

# Arrays in Java

## 1. Introduction to Arrays

An **array** is an **indexed, fixed-size collection of homogeneous elements**.

### Key Characteristics

- Stores **multiple values under a single variable name**
- Elements are accessed using **zero-based index**
- Size is **fixed at creation time**
- Arrays are **objects** in Java (stored on heap)

```
int[] a = new int[3];
```

---

### Advantages

- Improves **readability**
  - Efficient **random access** using index
  - Lower memory overhead compared to collections
- 

### Disadvantages (Exam-Relevant)

- **Fixed size** (cannot grow or shrink)
- Can cause **memory wastage or shortage**
- No built-in bounds checking at compile time

✓ These limitations are addressed using **Collections Framework**

---

## 2. Array Declarations

### Single-Dimensional Arrays

All the following are **valid**:

```
int[] a;    // recommended
int []a;
int a[];
```

**✗** Size cannot be specified during declaration:

```
int[5] a; // Compile-time error
```

---

### Multi-Dimensional Array Declarations

Java supports **any number of dimensions**.

#### 2D Arrays (6 valid forms)

```
int[][] a;
int []a[];
int[] []a;
int a[][];
```

#### 3D Arrays (10 valid forms)

```
int[][][] a;
int[] [][]a;
int[][] a[];
```

---

### Multiple Variables in One Declaration (Common Trap)

```
int[] a1, b1;    // both 1D arrays ✓
int[] a2[], b2;  // a2 = 2D, b2 = 1D ✓
int[] a, []b;    // ✗ invalid
```

Rule: **Dimension before variable name applies only to the first variable**

---

### 3. Array Construction

Arrays are created using the **new operator**, because arrays are objects.

```
int[] a = new int[3];
```

---

#### JVM-Level Array Classes (Conceptual)

Array Type	JVM Class
------------	-----------

<code>int[]</code>	<code>[I</code>
--------------------	-----------------

<code>int[][]</code>	<code>[[I</code>
----------------------	------------------

<code>double[]</code>	<code>[D</code>
-----------------------	-----------------

These classes exist at JVM level but are **not accessible directly**

---

#### Rules During Array Creation

##### Rule 1: Size is Mandatory

```
new int[]; // ❌ Compile-time error
```

---

##### Rule 2: Zero-Length Arrays Are Valid

```
int[] a = new int[0];  
System.out.println(a.length); // 0
```

✓ Very important for edge cases

---

##### Rule 3: Negative Size → Runtime Exception

```
new int[-3]; // NegativeArraySizeException
```

---

##### Rule 4: Allowed Types for Size

Only:

- `byte`
- `short`
- `char`
- `int`

```
new int['a']; // valid
new int[10L]; // ❌ CE
new int[10.5]; // ❌ CE
```

---

#### Rule 5: Maximum Array Size

- Maximum index range is `Integer.MAX_VALUE`
- Practically limited by **heap memory**

```
new int[2147483647]; // may cause OutOfMemoryError
```

---

## 4. Multi-Dimensional Arrays (Array of Arrays)

Java uses **array-of-arrays**, not matrix memory.

```
int[][] a = new int[2][];
a[0] = new int[3];
a[1] = new int[2];
```

✓ Each row can have **different length** (jagged arrays)

---

### 3D Example

```
int[][][] a = new int[2][][];
a[0] = new int[3][];
a[1] = new int[2][2];
```

- ✓ Improves **memory utilization**
- 

## 5. Array Initialization

### Default Values (Very High Exam Weight)

When an array is created, **all elements get default values**:

Type	Default
numeric	0 / 0.0
char	'\u0000'
boolean	false
reference	null

---

```
int[] a = new int[3];
System.out.println(a[0]); // 0
```

---

### Printing Arrays (Common Confusion)

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

Output:

```
[I@3e25a5
```

- ✓ Calls `toString()` → `class@hashcode`
- 

### Partial Initialization (Danger Zone)

```
int[][] a = new int[2][];
System.out.println(a[0]); // null
System.out.println(a[0][0]); // NullPointerException
```

---

## 6. Array Index Errors

```
a[4] = 10;    // ArrayIndexOutOfBoundsException
a[-1] = 20;   // ArrayIndexOutOfBoundsException
```

✓ Index must be:  $0 \leq \text{index} < \text{length}$

---

## 7. Declaration + Construction + Initialization (Single Line)

```
int[] a = {10,20,30};
String[] s = {"Java","Python"};
```

✓ Allowed **only in one statement**

✗ Invalid:

```
int[] x;
x = {10,20}; // Compile-time error
```

---

## Multi-Dimensional Shortcut

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

---

## 8. `length` vs `length()`

Feature	Applies To
<code>length</code>	Arrays
<code>length()</code>	String
<code>a.length;</code>	// ✓
<code>a.length();</code>	// ✗
<code>s.length();</code>	// ✓

```
s.length;    // ❌
```

---

## Multi-Dimensional Arrays

```
int[][] a = new int[6][3];  
a.length;    // 6 (base size)  
a[0].length; // 3
```

✓ No direct way to get total elements

---

## 9. Anonymous Arrays

Arrays **without a reference name**, used for **instant use**.

```
new int[] {10, 20, 30};
```

✓ Commonly used in method calls:

```
sum(new int[] {10, 20, 30});
```

❌ Size cannot be specified:

```
new int[3] {10, 20}; // CE
```

---

## 10. Array Element Assignments

### Case 1: Primitive Arrays

Allowed if **promotion is possible**:

```
int[] a = new int[5];  
a[0] = 'a';    // ✓  
a[1] = 10L;    // ❌
```

---

### Case 2: Object Type Arrays

Can store:

- Declared type
- Child class objects

```
Object[] o = new Object[3];  
o[0] = new String("Java");  
o[1] = new Integer(10);
```

---

### Case 3: Interface Type Arrays

```
Runnable[] r = new Runnable[2];  
r[0] = new Thread(); // ✓  
r[1] = new String(); // ✗
```

---

## 11. Array Variable Assignments (VERY IMPORTANT)

### Rule 1: No Element-Level Promotion at Array Level

```
char[] ch = {'a', 'b'};  
int[] i = ch; // ✗
```

---

### Rule 2: Reference Reassignment Only

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

- ✓ Elements not copied
  - ✓ Sizes need not match
  - ✓ Types must match
- 

### Rule 3: Dimensions Must Match

```
int[][] a = new int[3][];  
a[0] = new int[4]; // ✓  
a[0] = 10;          // ✗
```

---

## 12. Arrays and Garbage Collection (Advanced but Tested)

```
int[][] a = new int[3][2];  
a[0] = new int[3];  
a[1] = new int[4];  
a = new int[4][3];
```

- ✓ Old arrays become **eligible for GC**
  - ✓ Reassignment affects object reachability
- 

## 13. Enhanced **for** Loop with Arrays

```
for(int x : a) {  
    System.out.println(x);  
}
```

- ✓ No index
  - ✗ Cannot modify array structure
  - ✓ Safer, cleaner traversal
- 

## Certification Takeaways (Must Memorize)

- Arrays are objects
- Size is fixed
- Default values always assigned
- Multi-dimensional arrays are arrays of arrays
- `length`  $\neq$  `length()`
- Array assignment copies references, not elements

- Promotion rules differ for **elements vs arrays**
  - Index errors are runtime exceptions
  - Anonymous arrays are for **one-time use**
  - JVM class names (`[I`, `[[I`) appear in output
- 

## Modern Relevance

- Arrays are still used in:
  - Performance-critical code
  - JVM internals
  - Low-level libraries
- Most business logic uses **Collections**, but **Oracle exams focus heavily on arrays**