Mastering 1D Arrays

A Deep Dive into Java Data Structures

Memory Syntax Coding

Introduction

What are arrays and why do we need them?

What is an Array?

Definition

An array is a container object that holds a fixed number of values of a single type. It is a fundamental data structure in Java used to store a collection of data.

The Concept

Think of it as a single variable that can hold multiple values. Instead of declaring `num1`, `num2`, `num3...`, you declare one array `numbers` that holds them all.

Key Characteristics

Homogeneous

All elements within an array must be of the same data type (e.g., all integers or all strings).

Fixed Size

Once an array is created, its length cannot be changed. It is static in nature.

Indexed

Elements are accessed via a numerical index, starting strictly from 0 up to length-1.

Real Life Analogy: Lockers

Sequential Storage

Imagine a row of lockers in a hallway. Each locker has a specific, sequential number (index) and holds a student's items (value).

To get your books, you don't look through every locker; you go directly to locker #105 because you know the address.

Fixed Capacity

An egg carton is another perfect analogy for an array's "Fixed Size" property.

A standard carton has exactly 12 slots. You cannot magically add a 13th slot to it. If you need to store more eggs, you need a completely new, larger carton.

In Java, if you need more space, you must create a new array and copy the contents over.

Memory Structure

Understanding what happens under the hood.

Contiguous Memory Allocation

How it works

Arrays are stored in **contiguous** memory blocks. This means the system reserves a solid chunk of memory for the entire array.

Because the memory is continuous and the data types are the same size, Java can instantly calculate the address of any element using the index.

*Integer = 4 bytes, so address increases by 4.

The Golden Rule

> *"Arrays in Java are 0-indexed. The first element is always at index 0, and the last element is at length-1."*

Declaration Syntax

```
// Style 1 (Preferred - Java Standard)
int[] numbers;

// Style 2 (C-Style - Avoid)
int numbers[];
```

Declaration vs Creation

Declaring an array variable does **not** create the array in memory. It simply creates a reference variable that *can* point to an array.

At this stage, the value of the variable is `null`.

Instantiation

The new Keyword

To actually create the array container in memory, we must use the new keyword followed by the type and the size.

```
numbers = new int[5];
```

One-Liner

You can combine declaration and instantiation in a single line. This is the most common way to create an empty array.

```
int[] numbers = new int[5];
```

Initialization Methods

Static Initialization

Use when you know the values beforehand. No need to specify the size; Java infers it.

```
int[] primes = {2, 3, 5, 7};
```

Dynamic Initialization

Use when values will be calculated or input later. The array is filled with default values initially.

```
int[] marks = new int[10];
marks[0] = 95;
```

Default Values

When initialized with `new`, arrays are automatically filled:

`0`	`0.0`	`false`	`null`
int / byte / short	double / float	boolean	String / Object

Core Operations

Accessing, Modifying, and Iterating.

Accessing Elements

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

// Access the first element
int first = arr[0];

// Access the last element
int last = arr[2];
```

Index Notation

We use square brackets `[]` containing the index to retrieve a value.

Remember, accessing an index that doesn't exist (like `arr[3]` in this example) will cause a runtime error.

Modifying Elements

Assignment

You can change the value at any specific index using the assignment operator =.

The previous value is overwritten immediately. This operation is very fast (O(1) time complexity).

```
String[] fruits = {"Apple", "Banana"};

// Change Banana to Cherry
fruits[1] = "Cherry";

System.out.println(fruits[1]);
// Output: Cherry
```

Finding Array Size

The `.length` Property

Arrays have a built-in property called `length` that returns the total capacity of the array.

```
int size = arr.length;
```

⚠ Note: It is `.length` (no parentheses), unlike String's `.length()`.

Iterating: Standard For Loop

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

Full Control

The traditional for loop is ideal when you need to:

- Know the index of the current element.
- Modify elements while iterating.
- Iterate in reverse order or skip elements (e.g., i += 2).

Iterating: Enhanced For Loop

Read-Only / Sequential

Also known as the "For-Each" loop. It is cleaner and less error-prone.

Use this when you just need to read every element in order and don't care about the index.

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

Exception Handling

ArrayIndexOutOfBounds

This exception occurs when you try to access an index that is negative or greater than or equal to the array's size.

```
int[] arr = new int[5];
arr[5] = 10; // Error! Max index is 4
```

Null vs Empty Array

Null Array

```
int[] a = null;
```

The variable exists but points to nothing.
Accessing `a.length` throws NullPointerException.

Empty Array

```
int[] a = new int[0];
```

The object exists in memory, but has no slots.
`a.length` is 0. Safe to access length.

Code: Sum of Elements

```
int[] nums = {10, 20, 30};
int sum = 0;

for (int n : nums) {
  sum += n;
}
System.out.println("Sum: " + sum);
```

Logic

1. Initialize a variable `sum` to 0.
2. Iterate through each number in the array.
3. Add the current number to the running sum.
4. Print the result after the loop finishes.

Code: Find Maximum

Logic

1. Assume the first element (index 0) is the maximum.
2. Loop through the rest of the array starting from index 1.
3. If the current element is greater than our current max, update max.

```
int[] arr = {5, 12, 3, 9};
int max = arr[0];

for (int i = 1; i < arr.length; i++) {
  if (arr[i] > max) {
    max = arr[i];
  }
}
```

Code: Linear Search

```
int target = 30;
boolean found = false;

for (int n : arr) {
  if (n == target) {
    found = true;
    break;
  }
}
```

Logic

We want to check if a specific value exists in the array.

We iterate through every element. If we find a match, we set a flag to true and `break` (stop) the loop to save time.

Q & A

Thank you for learning!