Java Arrays - Complete Theory Notes

📚 Table of Contents

- Java Arrays - Complete Theory Notes
  - 📚 Table of Contents
  - Introduction to Arrays
    - Key Points:
  - Features of Arrays
    - When to Use Arrays:
  - Array Declaration
    - Syntax Options:
    - Examples:
  - Array Creation
    - 1. Without using `new` keyword
    - 2. Using `new` keyword
  - Array Operations
    - Reverse Array
    - Rotate Array
    - Frequency Count
    - Second Largest Element
  - Array Traversal
    - 1. For Loop
    - 2. Enhanced For Loop (For-Each)
  - Sorting
    - Bubble Sort
  - 💡 Coding Trick
  - 📝 Important Points to Remember
  - 🎯 Common Array Methods

Introduction to Arrays

Array is a group or collection of similar types of elements stored in contiguous memory locations.

Key Points

- Stores elements of the **same data type**
- **Fixed size** - once created, size cannot be increased or decreased
- **Zero-based indexing** - first element at index 0, last at index (length-1)
- Arrays are **non-primitive data types** (reference types) stored in **Heap memory**
- Provides **fast performance** due to direct index-based access
- Arrays are **objects** in Java

Features of Arrays

Feature	Description
Fixed Size	Once created, array size cannot be modified
Similar Type	Can only store elements of the same data type
Zero Indexing	Index starts from 0 and ends at (length - 1)
No Built-in Methods	Limited built-in methods compared to Collections
Variable Length Property	Has `.length` property to get array size
Fast Access	Direct index-based access provides O(1) performance
Memory Management	Not efficient in memory management due to fixed size
Location	Stored in Heap area as objects

When to Use Arrays

- Use arrays when you know the **exact size** beforehand
- When you need **fast access** to elements by index
- When performance is critical
- Avoid when size is dynamic or frequent insertions/deletions are needed

---

Array Declaration

Syntax Options

```
// Option 1: Brackets after data type (Recommended)
dataType[] varName;

// Option 2: Brackets after variable name
dataType varName[];
```

Examples

```
// Integer array
int[] numbers;

// Double array
double[] salaries;

// String array
String[] names;

// Boolean array
boolean[] isPresent;

// Character array
char[] characters;

// Employee array (custom object)
Employee[] employees;
```

Array Creation

Arrays can be created in **two ways**:

1. Without using `new` keyword

Array Literal Syntax:

```
dataType[] varName = {value1, value2, value3, ...};
```

Examples:

```java
// Integer array with 5 elements
int[] nums = {10, 20, 30, 40, 50};
System.out.println(nums.length);  // Output: 5

// Accessing elements
System.out.println(nums[0]);                // Output: 10 (first element)
System.out.println(nums[2]);                // Output: 30 (third element)
System.out.println(nums[nums.length - 1]);  // Output: 50 (last element)

// ⚠️ ArrayIndexOutOfBoundsException
// System.out.println(nums[nums.length]);   // Error! Index 5 doesn't exist

// Multiple examples
int[] a = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
double[] prices = {99.99, 149.50, 79.99};
String[] cities = {"Delhi", "Mumbai", "Bangalore"};
```

Memory Representation:

```
Stack Area:
+-------+
| nums  | ──────┐
+-------+       |
                |
                ▼
Heap Area:
Index:  [0] [1] [2] [3] [4]
Value:  [10][20][30][40][50]
```

---

2. Using `new` keyword

Syntax:

```java
dataType[] varName = new dataType[size];
```

Examples:

```java
// Create integer array of size 5
int[] nums = new int[5];

// Assign values individually
nums[0] = 12;
nums[1] = 25;
nums[2] = 48;
nums[3] = 36;
nums[4] = 89;

// Create and initialize in one line
int[] a = new int[]{3, 5, 200};

// Default values for different types
int[] defaultInt = new int[5];
// All elements initialized to 0: [0, 0, 0, 0, 0]

double[] defaultDouble = new double[3];
// All elements initialized to 0.0: [0.0, 0.0, 0.0]

boolean[] defaultBoolean = new boolean[3];
// All elements initialized to false: [false, false, false]

String[] names = new String[3];
// All elements initialized to null: [null, null, null]
```

Default Values:

Data Type	Default Value
int, byte, short, long	0
float, double	0.0
boolean	false
char	'\u0000' (null character)
Reference types (String, Object, etc.)	null

Array Operations

Reverse Array

Problem: Reverse the elements of an array

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

// Using two-pointer approach
int left = 0;
int right = nums.length - 1;

while (left < right) {
    // Swap elements
    int temp = nums[left];
    nums[left] = nums[right];
    nums[right] = temp;

    left++;
    right--;
}

// Result: [50, 40, 30, 20, 10]
```

Rotate Array

Problem: Rotate array by k positions

```
// Rotate right by 2 positions
// Input: [1, 2, 3, 4, 5], k = 2
// Output: [4, 5, 1, 2, 3]

public static void rotateArray(int[] arr, int k) {
    int n = arr.length;
    k = k % n; // Handle k > n

    // Reverse entire array
    reverse(arr, 0, n - 1);
    // Reverse first k elements
    reverse(arr, 0, k - 1);
    // Reverse remaining elements
    reverse(arr, k, n - 1);
}

private static void reverse(int[] arr, int start, int end) {
    while (start < end) {
        int temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;
        start++;
        end--;
    }
}
```

Frequency Count

Problem: Count frequency of each element

```
int[] nums = {1, 2, 2, 3, 3, 3, 4, 4, 4, 4};

// Using array to store frequency
int[] frequency = new int[100]; // Assuming max value < 100

for (int num : nums) {
    frequency[num]++;
}

// Print frequencies
for (int i = 0; i < frequency.length; i++) {
    if (frequency[i] > 0) {
        System.out.println(i + " appears " + frequency[i] + " times");
    }
}
```

Second Largest Element

Problem: Find the second largest element in array

```
int[] nums = {10, 50, 30, 80, 70};

int largest = Integer.MIN_VALUE;
int secondLargest = Integer.MIN_VALUE;

for (int num : nums) {
    if (num > largest) {
        secondLargest = largest;
        largest = num;
    } else if (num > secondLargest && num != largest) {
        secondLargest = num;
    }
}

System.out.println("Second Largest: " + secondLargest); // Output: 70
```

Array Traversal

1. For Loop

Traditional for loop with index:

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

// Forward traversal
for (int i = 0; i < nums.length; i++) {
    System.out.println(nums[i]);
}

// Output:
// 10
// 20
// 30
// 40
// 50

// Backward traversal
for (int i = nums.length - 1; i >= 0; i--) {
    System.out.println(nums[i]);
}
```

Advantages:

- Can access index
- Can modify elements
- Can traverse in any direction

---

2. Enhanced For Loop (For-Each)

Syntax:

```java
for (dataType varName : array) {
    // code
}
```

Examples:

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

// For-each loop
for (int num : nums) {
    System.out.println(num);
}

// With Strings
String[] names = {"Alice", "Bob", "Charlie"};
for (String name : names) {
    System.out.println("Hello, " + name);
}
```

Advantages:

- Cleaner and more readable
- No index management

- Less prone to errors

Limitations:

- Cannot access index
- Cannot modify array elements
- Only forward traversal

Sorting

Bubble Sort

Algorithm: Compare adjacent elements and swap if in wrong order

Implementation:

```
public static void bubbleSort(int[] arr) {
    int n = arr.length;

    for (int i = 0; i < n - 1; i++) {
        boolean swapped = false;

        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // Swap
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                swapped = true;
            }
        }

        // If no swaps, array is sorted
        if (!swapped) break;
    }
}

// Example usage
int[] nums = {64, 34, 25, 12, 22, 11, 90};
bubbleSort(nums);
// Result: [11, 12, 22, 25, 34, 64, 90]
```

Time Complexity:

- Best Case: O(n) - when array is already sorted
- Average Case: O(n²)
- Worst Case: O(n²)

Space Complexity: O(1)

💡 Coding Trick

Print even/odd without if-else or ternary operator:

```java
int n = 325;
String[] s = {"Even", "Odd"};
System.out.println(n + " is: " + s[n % 2]);

// Explanation:
// n % 2 = 0 (even) → s[0] → "Even"
// n % 2 = 1 (odd)  → s[1] → "Odd"
```

📝 Important Points to Remember

1. **Array Index:** Always starts from 0
2. **Array Length:** Use `array.length` (not `array.length()`)
3. **Fixed Size:** Cannot resize after creation
4. **Default Values:** Numeric types → 0, boolean → false, objects → null
5. **Exception:** `ArrayIndexOutOfBoundsException` when accessing invalid index
6. **Memory:** Arrays are stored in Heap, references in Stack
7. **Performance:** O(1) for access, but O(n) for search in unsorted array
8. **Copy:** Use `System.arraycopy()` or `Arrays.copyOf()` for copying

🎯 Common Array Methods

```java
import java.util.Arrays;

int[] nums = {5, 2, 8, 1, 9};

// Sort array
Arrays.sort(nums); // [1, 2, 5, 8, 9]

// Binary search (array must be sorted)
int index = Arrays.binarySearch(nums, 5); // Returns index

// Fill array with value
Arrays.fill(nums, 0); // All elements become 0

// Copy array
int[] copy = Arrays.copyOf(nums, nums.length);

// Compare arrays
boolean isEqual = Arrays.equals(nums, copy);

// Convert to String
String str = Arrays.toString(nums); // "[1, 2, 5, 8, 9]"
```

End of Notes ✨