



Ahmad Albarasy · Follow

Last edited by Ahmad Albarasy on Nov 15, 2025

Contributed by



DsaA Lab 1 - Arrays

The Islamic University of Gaza
Engineering Faculty
Department of Computer Engineering

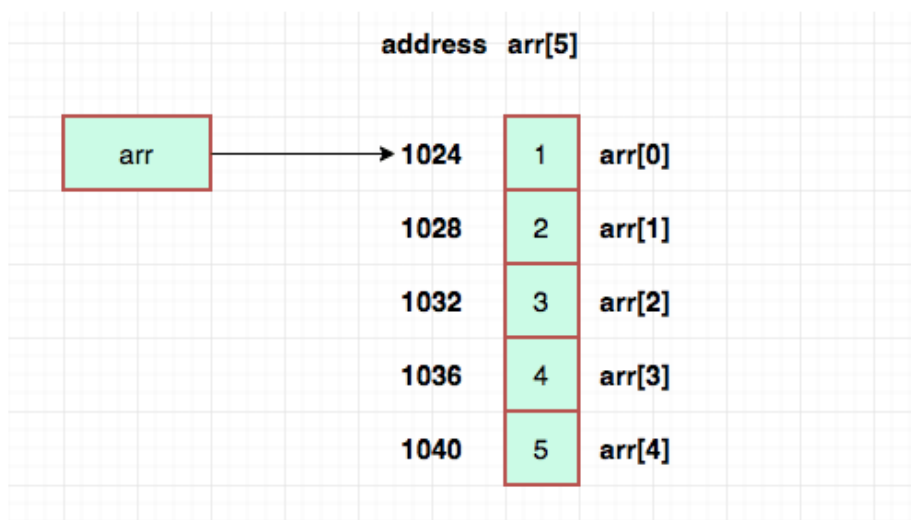
Prepared by: Eng. Ahmad Albarasy

Under the supervision of: Prof. Ahmad Mahdi

In this lab, we are going to dive into arrays, one of the most basic yet fundamental data structures used in almost every peice of code we write. It serves as a ground base for implementing many more complex data structures that we will explore in later labs. But for now, lets get started!

Definition & Usage

An array is a linear data structure that has **a fixed size** and stores elements of the same type (or memory size) in contiguous locations in memory. An array is stored such that the position (memory address) of each element can be computed from the base index using a simple mathematical formula.



For instance, if we have an array that has a length of 5, and it stores elements of type `int`.

Declaration and Initialization

To create an array in Java, we have to declare it first then we can initialize it with our size of choice:

```
1 | int[] arr;  
2 | arr = new int[5];
```

We can both declare and initialize the array in one step:

```
1 | int[] arr = new int[5];
```

Alternatively, we can directly specify the elements of the array, which also defines the size that is going to be used:

```
1 | int[] arr = { 1, 2, 3, 4, 5 }; // arr.length = 5
```

Accessing and Modifying Elements

To access the first element, we have to use index `0`, since Java is a zero-based indexing language (just like almost every other programming language):

```
1 | int firstInt = arr[0];
```

intuitively, to access the last element, we use the index `n-1`, where `n` is the length of the array:

```
1 | int lastInt = arr[4];
```

In Java, if we attempt to access an index that are out of the array's boundaries, Java will throw an `ArrayIndexOutOfBoundsException` in runtime, resulting in crashing your program.

To modify an element, we can simply assign a value the index we are willing to modify:

```
1 | arr[4] = 10; // our array becomes [1, 2, 3, 4, 10]
```

A side note

The size of an integer varies depending on the programming language. Common integer sizes include 16-bit, 32-bit, and 64-bit. **In Java**, however, **the `int` type always occupies 4 bytes (32 bits)**, regardless of whether the system is 32-bit or

64-bit. This size is guaranteed across all Java Virtual Machine (JVM) implementations and platforms.

Iterating through an array

Now, let's say that we want to iterate over the array and print its values. To do so, we use a simple for loop that starts from index 0, and exits when the condition `index < length` is not met

```
1 public class Main {  
2     public static void main(String[] args){  
3         int[] arr = { 1, 2, 3, 4, 5 };  
4         for(int i = 0; i < arr.length; i++){  
5             System.out.println(arr[i]);  
6         }  
7     }  
8 }
```

Types of Arrays

In terms of dimensions, arrays have 2 types:

1. One Dimensional Array

A one dimensional array is, as the name implies, an array that has only one dimension to it, and it can be accessed using a single index. It is commonly used to store a collection of elements of the same type in an ordered sequence and perform operations on them.

```
1 short[] grades = { 60, 70, 80 };
```

2. Multi-Dimensional Array

A multi-dimensional array is an array that has more than one dimension. It can be thought of as an array of arrays, where each element of the main array is itself an array.

```
1 int[][] unityMatrix = {  
2     {1, 0, 0},  
3     {0, 1, 0},  
4     {0, 0, 1}  
5 }
```

The most common type is the two-dimensional array, which can be visualized as a table or a grid with rows and columns. Multi-dimensional arrays are useful for representing matrices, game boards, or any data that naturally fits into a tabular structure.

Real World Examples

Arrays are pretty much everywhere, you definitely have already used them in your previous programming course last year, but here are some real-world examples just for the sake of lab.

Mapping indices to objects/values

It can be as simple as this, using an array to map an index to a specific value. For example, you can map the indices from `0` to `11` to the months of the year:

```
1 | String[] months = {  
2 |     "Jan", "Feb", "Mar", "Apr", "May", "Jun",  
3 |     "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"  
4 | };
```

Another example would be mapping a vector of 3 elements representing an RGB color:

```
1 | short[] color = {255, 0, 255} // magenta
```

Matrix Multiplication

One of the practical uses of arrays, especially multi-dimensional ones, is to perform matrix multiplication. Take a look at this example, which multiplies a 2×3 matrix with a 3×2 one, resulting in a 2×2 resultant matrix:

```
1 public class MatrixMultiplication {
2     public static void main(String[] args) {
3         int[][] A = {
4             { 1, 2, 3 },
5             { 4, 5, 6 }
6         };
7
8         int[][] B = {
9             { 7, 8 },
10            { 9, 10 },
11            { 11, 12 }
12        };
13
14        int[][] result = multiplyMatrices(A, B);
15
16        System.out.println("Resultant Matrix:");
17        printMatrix(result);
18    }
19
20    public static int[][] multiplyMatrices(int[][] A, int[][] B) {
21        int rowsA = A.length;
22        int colsA = A[0].length;
23        int colsB = B[0].length;
24
25        int[][] result = new int[rowsA][colsB];
26
27        for (int i = 0; i < rowsA; i++) {
28            for (int j = 0; j < colsB; j++) {
29                for (int k = 0; k < colsA; k++) {
30                    result[i][j] += A[i][k] * B[k][j];
31                }
32            }
33        }
34
35        return result;
36    }
37
38    public static void printMatrix(int[][] matrix) {
39        for (int[] row : matrix) {
40            for (int value : row) {
41                System.out.print(value + "\t");
42            }
43            System.out.println();
44        }
45    }
46 }
```

Matrix multiplication is used in a variety of applications, including image processing, graphics, video games, cheating in your linear algebra homework, physics simulations, computer vision, and machine learning algorithms such as neural networks

Tasks

Task 1 (4 points)

Write a program that reverses the order of numbers in an array in-place and prints out the result.

Use the following code to implement and submit your solution:

```
1  /* make sure to add this package identifier before you
2   * submit your solution to GradeScope. Remove it while testing locally */
3  package com.gradescope.labOne;
4
5  public class TaskOne {
6      public static void reverse(int[] arr){
7          // implement your solution here
8      }
9
10 }
```

You can use the following code to test your solution:

```

1  import java.util.Arrays;
2
3  public class TaskOneTests {
4      public static void main(String[] args) {
5          /* ----- Test Case 1 ----- */
6          int[] arr1 = {1, 2, 3, 4, 5, 6};
7          System.out.println("Original Array: " + Arrays.toString(arr1));
8          TaskOne.reverse(arr1);
9          System.out.println("Reversed Version: " + Arrays.toString(arr1));
10         System.out.println("Expected: [6, 5, 4, 3, 2, 1]\n");
11
12         /* ----- Test Case 2 ----- */
13         int[] arr2 = {5, 4, 3, 2, 1};
14         System.out.println("Original Array: " + Arrays.toString(arr2));
15         TaskOne.reverse(arr2);
16         System.out.println("Reversed Version: " + Arrays.toString(arr2));
17         System.out.println("Expected: [1, 2, 3, 4, 5]\n");
18
19         /* ----- Test Case 3 ----- */
20         int[] arr3 = {42};
21         System.out.println("Original Array: " + Arrays.toString(arr3));
22         TaskOne.reverse(arr3);
23         System.out.println("Reversed Version: " + Arrays.toString(arr3));
24         System.out.println("Expected: [42]\n");
25
26         /* ----- Test Case 4 ----- */
27         int[] arr4 = {};
28         System.out.println("Original Array: " + Arrays.toString(arr4));
29         TaskOne.reverse(arr4);
30         System.out.println("Reversed Version: " + Arrays.toString(arr4));
31         System.out.println("Expected: []\n");
32
33         /* ----- Test Case 5 ----- */
34         int[] arr5 = {10, -5, 0, 8};
35         System.out.println("Original Array: " + Arrays.toString(arr5));
36         TaskOne.reverse(arr5);
37         System.out.println("Reversed Version: " + Arrays.toString(arr5));
38         System.out.println("Expected: [8, 0, -5, 10]\n");
39     }
40 }

```

Task 2 (3 points)

You are given an array of integers representing daily wind speeds in km/h.

A **calm period** is defined as a sequence of consecutive days where the wind speed never exceeds a certain threshold τ

write a program that finds the longest calm period given an array representing the wind speeds over 15 days.



Tip

Take a look at the sliding window approach, which is a common technique to solve some array-related problems. Give it a quick search :>

Use the following code to implement and submit your solution:

```

1  /* make sure to add this package identifier before you
2   * submit your solution to GradeScope. Remove it while testing locally */
3  package com.gradescope.labOne;
4
5  public class TaskTwo {
6
7      public static int longestCalmPeriod(int[] speeds, int T) {
8          // implement your solution here
9          return 0; // this is just a placeholder
10     }
11 }
```

You can use the following code to test your solution:

```

1  public class TaskTwoTests {
2      public static void main(String[] args) {
3          /* ----- Test Case 1 ----- */
4          int[] speeds1 = { 5, 8, 9, 3, 2, 1, 11, 4, 5, 13, 16, 14, 7, 2, 4 };
5          int T1 = 8;
6          int result1 = TaskTwo.longestCalmPeriod(speeds1, T1);
7          System.out.println("Test Case 1:");
8          System.out.println("Threshold: " + T1);
9          System.out.println("Expected Output: 3");
10         System.out.println("Your Output: " + result1);
11         System.out.println();
12
13         /* ----- Test Case 2 ----- */
14         int[] speeds2 = { 10, 7, 6, 5, 12, 9, 3, 3, 4, 14, 4, 8, 8, 5, 7 };
15         int T2 = 8;
16         int result2 = TaskTwo.longestCalmPeriod(speeds2, T2);
17         System.out.println("Test Case 2:");
18         System.out.println("Threshold: " + T2);
19         System.out.println("Expected Output: 5");
20         System.out.println("Your Output: " + result2);
21     }
22 }
```

Task 3 (3 points)

Given an array of integers `nums` and an integer `target`, return indices of the two numbers such that they add up to `target`.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

You can return the answer in any order.

Question Link on [LeetCode](#)

Example 1:

Input: nums = [2,7,11,15] , target = 9

Output: [0,1]

Explanation: Because $\text{nums}[0] + \text{nums}[1] == 9$, we return [0, 1] .

Example 2:

Input: nums = [3,2,4] , target = 6

Output: [1,2]

Example 3:

Input: nums = [3,3] , target = 6

Output: [0,1]

Use the following code to implement and submit your solution:

```
1  /* make sure to add this package identifier before you
2   * submit your solution to GradeScope. Remove it while testing locally */
3  package com.gradescope.labOne;
4
5  public class TaskThree {
6
7      public static int[] twoSum(int[] nums, int target) {
8          // implement your solution here
9      }
10 }
11
```

Use the following code to test your solution. Alternatively, you can solve it on LeetCode to get used to it :)


```
1  import java.util.Arrays;
2
3  public class TaskThreeTests {
4      public static void main(String[] args) {
5          testSimpleCase();
6          testPairAtEnds();
7          testDuplicateValues();
8          testNegativeNumbers();
9      }
10
11     public static void testSimpleCase() {
12         int[] nums = {2, 7, 11, 15};
13         int target = 9;
14         int[] result = TaskThree.twoSum(nums, target);
15         int[] expected = {0, 1};
16
17         System.out.println("Test 1 - Simple Case");
18         printResult(result, expected);
19     }
20
21     public static void testPairAtEnds() {
22         int[] nums = {1, 2, 3, 4, 10};
23         int target = 11;
24         int[] result = TaskThree.twoSum(nums, target);
25         int[] expected = {0, 4};
26
27         System.out.println("Test 2 - Pair at Ends");
28         printResult(result, expected);
29     }
30
31     public static void testDuplicateValues() {
32         int[] nums = {3, 3, 4, 5};
33         int target = 6;
34         int[] result = TaskThree.twoSum(nums, target);
35         int[] expected = {0, 1};
36
37         System.out.println("Test 3 - Duplicate Values");
38         printResult(result, expected);
39     }
40
41     public static void testNegativeNumbers() {
42         int[] nums = {-1, 5, 1, 2};
43         int target = 1;
44         int[] result = TaskThree.twoSum(nums, target);
45         int[] expected = {0, 3};
46
47         System.out.println("Test 4 - Negative Numbers");
48         printResult(result, expected);
49     }
50
51     private static void printResult(int[] result, int[] expected) {
52         System.out.println("Expected: " + (expected == null ? "null" : Arrays.toString(expected)));
53         System.out.println("Actual:    " + (result == null ? "null" : Arrays.toString(result)));
54         System.out.println("-----");
55     }
56 }
57
58
```

Happy Coding 😊

Last changed by

Ahmad Albarasy · Follow

0



209



Read more

Data Structures and Algorithms Lab 2 - Linked Lists

In this lab, we are going to learn about Linked Lists. What they are, how can we implement them, their key...

Nov 10, 2025

Data Structures and Algorithms Lab 0 - Introduction

Welcome one and all to the Data Structures and Algorithms lab! This lab is one of the most important...

Nov 8, 2025

Read more from Ahmad Albarasy

Published on  HackMD