Arrays in Java: One-Dimensional and Multi-Dimensional

Premanand S / SENSE / VIT-CC

February 3, 2025

Introduction to Arrays

- An array is a collection of elements of the same data type.
- Arrays are stored in contiguous memory locations.
- Indexing starts from 0 (zero-based indexing).
- Types:
  - One-dimensional arrays: A simple list of elements.
  - Multi-dimensional arrays: Arrays of arrays (e.g., 2D, 3D).

One-Dimensional Arrays

- Declaration and Initialization:

```java
// Declaration
int[] arr;  // Preferred way
int arr[];  // Alternative syntax

// Initialization with size
arr = new int[5];  // Creates an array of size 5,
    default values are 0 for int

// Initialization with values
int[] arr = {1, 2, 3, 4, 5};  // Creates an array
    with specific values

```

One-Dimensional Arrays

- Accessing Elements:

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

System.out.println(arr[0]);  // Output: 10
System.out.println(arr[2]);  // Output: 30

// Modifying an element
arr[1] = 25;
System.out.println(arr[1]);  // Output: 25

```

One-Dimensional Arrays

- Iterating Over an Array:

```java
for (int i = 0; i < arr.length; i++) {
    System.out.println("Element at index " + i + ": " + arr[i]);
}

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

```

Indexing

1. Arrays only support zero-based indexing, meaning that the first element of an array is accessed using the index 0, the second element with index 1, and so on.
2. Negative indexing is not supported in Java arrays.

Zero-Based Indexing Example:

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

System.out.println(arr[0]);  // Output: 10 (first
    element)
System.out.println(arr[1]);  // Output: 20 (second
    element)

```

Negative Indexing Example:

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

// Attempting to access a negative index
System.out.println(arr[-1]);  // This will throw
    ArrayIndexOutOfBoundsException

```

Why Negative Indexing is Not Supported?

1. Java follows the C-style array indexing where arrays are zero-indexed. The language does not provide built-in support for negative indexing.
2. Arrays in Java are stored in contiguous memory locations, and the index is used as an offset from the base address of the array. Negative indices would imply accessing memory outside the bounds of the array, which is not allowed.

Simulating Negative Indexing

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

int getIndex(int[] array, int index) {
    if (index < 0) {
        // Convert negative index to positive index
        return array[array.length + index];
    } else {
        return array[index];
    }
}

// Accessing elements using both positive and "
    negative" indices
System.out.println(getIndex(arr, 1));
System.out.println(getIndex(arr, -1));
System.out.println(getIndex(arr, -2));

```

Practice Questions (1D Arrays)

1. Find the sum of all elements in an array.
2. Find the largest and smallest number in an array.
3. Reverse the elements of an array without using another array.
4. Check if a given array contains a specific element.
5. Sort the elements of an array in ascending order.

Solution: Sum of All Elements in a 1D Array

```java
public class SumArray {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        int sum = 0;
        for (int num : arr) {
            sum += num;
        }
        System.out.println("Sum: " + sum);
    }
}

```

Solution: Largest and Smallest Number in a 1D Array

```java
public class MinMaxArray {
    public static void main(String[] args) {
        int[] arr = {10, 20, 5, 40, 30};
        int min = arr[0], max = arr[0];
        for (int num : arr) {
            if (num < min) min = num;
            if (num > max) max = num;
        }
        System.out.println("Min: " + min + ", Max: " +
     max);
    }
}

```

Solution: Reverse an Array Without Using Another Array

```java
public class ReverseArray {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        int n = arr.length;
        for (int i = 0; i < n / 2; i++) {
            int temp = arr[i];
            arr[i] = arr[n - i - 1];
            arr[n - i - 1] = temp;
        }
        System.out.println("Reversed Array:");
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }
}

```

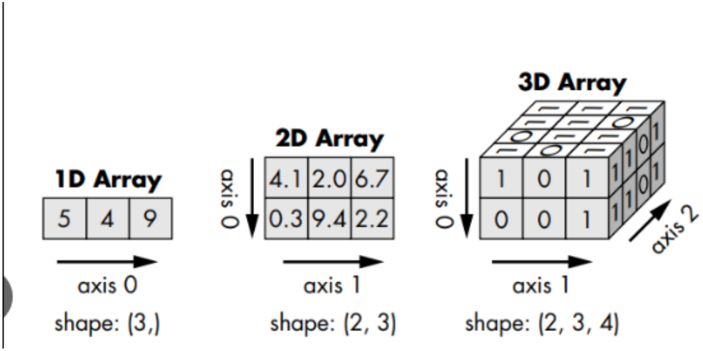
Solution: Check if an Array Contains a Specific Element

```java
public class ContainsElement {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        int target = 3;
        boolean found = false;
        for (int num : arr) {
            if (num == target) {
                found = true;
                break;
            }
        }
        System.out.println("Contains " + target + "? " + found);
    }
}
```

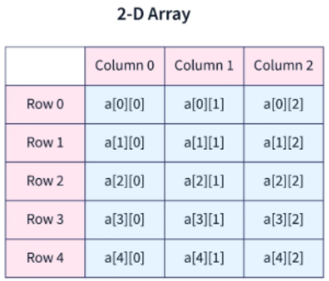
Solution: Sort an Array in Ascending Order

```java
import java.util.Arrays;

public class SortArray {
    public static void main(String[] args) {
        int[] arr = {5, 2, 9, 1, 5, 6};
        Arrays.sort(arr); // Built-in sort method
        System.out.println("Sorted Array:");
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }
}

```

1D Vs 2D Vs 3D

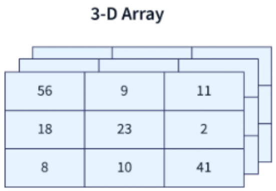

1D Array
5 4 9
axis 0
shape: (3,)
2D Array
4.1 2.0 6.7
0.3 9.4 2.2
axis 0
axis 1
shape: (2, 3)
3D Array
1 0 1
0 0 1
axis 0
axis 1
axis 2
shape: (2, 3, 4)

2D Vs 3D



2-D Array

	Column 0	Column 1	Column 2
Row 0	a[0][0]	a[0][1]	a[0][2]
Row 1	a[1][0]	a[1][1]	a[1][2]
Row 2	a[2][0]	a[2][1]	a[2][2]
Row 3	a[3][0]	a[3][1]	a[3][2]
Row 4	a[4][0]	a[4][1]	a[4][2]



3-D Array

56	9	11
18	23	2
8	10	41

Multi-Dimensional Arrays

1. Multi-dimensional arrays are arrays of arrays.
2. The most common type is a two-dimensional (2D) array, which can be visualized as a table or matrix with rows and columns.

Multi-Dimensional Arrays

- Declaration and Initialization:

```java
// Declaration
int[][] matrix;

// Initialization with size
matrix = new int[3][4];  // Creates a 3x4 matrix
    (3 rows, 4 columns)

// Initialization with values
int[][] matrix = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};

```

Multi-Dimensional Arrays

- Accessing Elements:

```java
int[][] matrix = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};

System.out.println(matrix[0][0]);  // Output: 1 (
    first row, first column)
System.out.println(matrix[1][2]);  // Output: 6 (
    second row, third column)

// Modifying an element
matrix[2][1] = 15;
System.out.println(matrix[2][1]);  // Output: 15

```

Iterating Through a 2D Array

```java
for (int i = 0; i < matrix.length; i++) {  // Iterate over rows
    for (int j = 0; j < matrix[i].length; j++) {  // Iterate over columns
        System.out.print(matrix[i][j] + " ");
    }
    System.out.println();  // New line after each row
}

// Enhanced for loop (for-each) for 2D array
for (int[] row : matrix) {
    for (int element : row) {
        System.out.print(element + " ");
    }
    System.out.println();
}

```

Practice Questions (2D Arrays)

1. Find the sum of all elements in a 2D array.
2. Find the transpose of a given 2D matrix.
3. Check if a given 2D array is a magic square.
4. Multiply two matrices.
5. Print the elements of a 2D array in spiral order.

Solution: Sum of All Elements in a 2D Array

```java
public class Sum2DArray {
    public static void main(String[] args) {
        int[][] matrix = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };
        int sum = 0;
        for (int[] row : matrix) {
            for (int num : row) {
                sum += num;
            }
        }
        System.out.println("Sum: " + sum);
    }
}

```

Solution: Transpose of a 2D Matrix

```java
public class TransposeMatrix {
    public static void main(String[] args) {
        int[][] matrix = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };
        int rows = matrix.length, cols = matrix[0].
    length;
        int[][] transpose = new int[cols][rows];
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                transpose[j][i] = matrix[i][j];
            }
        }
        System.out.println("Transposed Matrix:");
        for (int[] row : transpose) {
            for (int num : row) {
                System.out.print(num + " ");
            }
            System.out.println();
        }     } }
```

Solution: Check if a 2D Array is a Magic Square

```java
public class MagicSquare {
    public static void main(String[] args) {
        int[][] matrix = {
            {2, 7, 6},
            {9, 5, 1},
            {4, 3, 8}
        };
        int n = matrix.length;
        int sum = 0;
        for (int num : matrix[0]) sum += num; // First
     row sum
        boolean isMagic = true;

        // Check rows and columns
        for (int i = 0; i < n; i++) {
            int rowSum = 0, colSum = 0;
            for (int j = 0; j < n; j++) {
                rowSum += matrix[i][j];
                colSum += matrix[j][i];
            }


```

Solution: Check if a 2D Array is a Magic Square (Contd...)

```

            if (rowSum != sum || colSum != sum) {
                isMagic = false;
                break;
            }
        }

        // Check diagonals
        int diag1 = 0, diag2 = 0;
        for (int i = 0; i < n; i++) {
            diag1 += matrix[i][i];
            diag2 += matrix[i][n - i - 1];
        }
        if (diag1 != sum || diag2 != sum) isMagic =
    false;

        System.out.println("Is Magic Square? " +
    isMagic);
    } }

```

Solution: Multiply Two Matrices

```java
public class MatrixMultiplication {
    public static void main(String[] args) {
        int[][] A = {{1, 2}, {3, 4}};
        int[][] B = {{5, 6}, {7, 8}};
        int rowsA = A.length, colsA = A[0].length;
        int colsB = B[0].length;
        int[][] result = new int[rowsA][colsB];

        for (int i = 0; i < rowsA; i++) {
            for (int j = 0; j < colsB; j++) {
                for (int k = 0; k < colsA; k++) {
                    result[i][j] += A[i][k] * B[k][j];
                }              }         }

        System.out.println("Resultant Matrix:");
        for (int[] row : result) {
            for (int num : row) {
                System.out.print(num + " ");
            }
            System.out.println();
        }     }}

```

Solution: Print a 2D Array in Spiral Order

```java
public class SpiralOrder {
    public static void main(String[] args) {
        int[][] matrix = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };
        int top = 0, bottom = matrix.length - 1;
        int left = 0, right = matrix[0].length - 1;

        while (top <= bottom && left <= right) {
            // Traverse from Left to Right
            for (int i = left; i <= right; i++) {
                System.out.print(matrix[top][i] + " ")
    ;
            }
            top++;

```

Solution: Print a 2D Array in Spiral Order (Contd...)

```
            // Traverse Downwards
            for (int i = top; i <= bottom; i++) {
                System.out.print(matrix[i][right] + " ");
            }
            right--;
            // Traverse from Right to Left
            if (top <= bottom) {
                for (int i = right; i >= left; i--) {
                    System.out.print(matrix[bottom][i] + " ");
                }
                bottom--;
            }
            // Traverse Upwards
            if (left <= right) {
                for (int i = bottom; i >= top; i--) {
                    System.out.print(matrix[i][left] + " ");
                }
                left++;
            }          }      } }
```

Higher Dimensional Arrays (3D, 4D, etc.)

1. Java also supports higher-dimensional arrays like 3D, 4D, etc., though they are less commonly used.
2. A 3D array can be thought of as an array of 2D arrays.

Example of a 3D Array:

```java
// Declaration and initialization
int[][][] threeDArray = new int[2][3][4];

// Assigning values
threeDArray[0][0][0] = 1;
threeDArray[0][1][2] = 5;
threeDArray[1][2][3] = 9;

// Iterating over a 3D array
for (int i = 0; i < threeDArray.length; i++) {
    for (int j = 0; j < threeDArray[i].length; j++) {
        for (int k = 0; k < threeDArray[i][j].length; k++) {
            System.out.println("Element at (" + i + ", " + j + "," + k + "): " + threeDArray[i][j][k]);
        }
    }
}
```

Jagged Arrays

1. In Java, you can also create jagged arrays , where each row can have a different number of columns.

Jagged Arrays:

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

jaggedArray[0] = new int[2];  // First row has 2
    columns
jaggedArray[1] = new int[3];  // Second row has 3
    columns
jaggedArray[2] = new int[4];  // Third row has 4
    columns

// Assigning values
jaggedArray[0][0] = 1;
jaggedArray[0][1] = 2;
jaggedArray[1][0] = 3;
jaggedArray[1][1] = 4;
jaggedArray[1][2] = 5;
jaggedArray[2][0] = 6;
jaggedArray[2][1] = 7;
jaggedArray[2][2] = 8;
jaggedArray[2][3] = 9;

```

Jagged Arrays: (Contd...)

```java

// Printing jagged array
for (int i = 0; i < jaggedArray.length; i++) {
    for (int j = 0; j < jaggedArray[i].length; j
   ++) {
        System.out.print(jaggedArray[i][j] + " ");
    }
    System.out.println();
}

```

Common Operations on Arrays:

- Length : Use array.length for 1D arrays, array[i].length for 2D arrays, etc.
- Copying Arrays : Use System.arraycopy() or Arrays.copyOf().
- Sorting : Use Arrays.sort() to sort arrays.
- Searching : Use Arrays.binarySearch() for sorted arrays.

Keypoints

- Arrays are fundamental for storing and manipulating collections of data.
- One-dimensional arrays are simple lists of elements.
- Multi-dimensional arrays are arrays of arrays, typically used for matrices or tables.
- Jagged arrays allow rows to have different lengths.
- Higher-dimensional arrays (3D, 4D, etc.) are possible but less common.