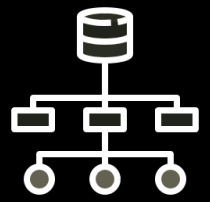
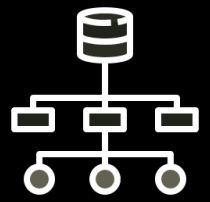
Engineer With SR



Array

Data Structures

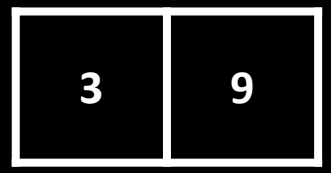


Agenda

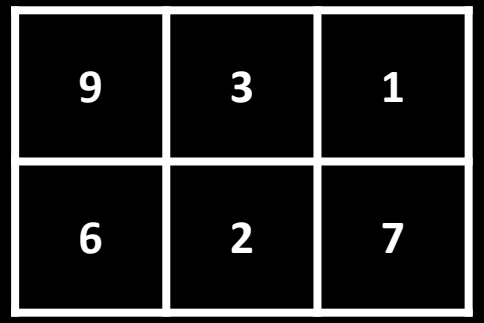
- Introduction
- Real-World Analogy
- Types
- Syntax
- Features / Characteristics
- Operations
- Example
- Algorithm
- Implementation
- Use Cases
- Applications
- Limitations
- Complexities
- Exceptions

Introduction

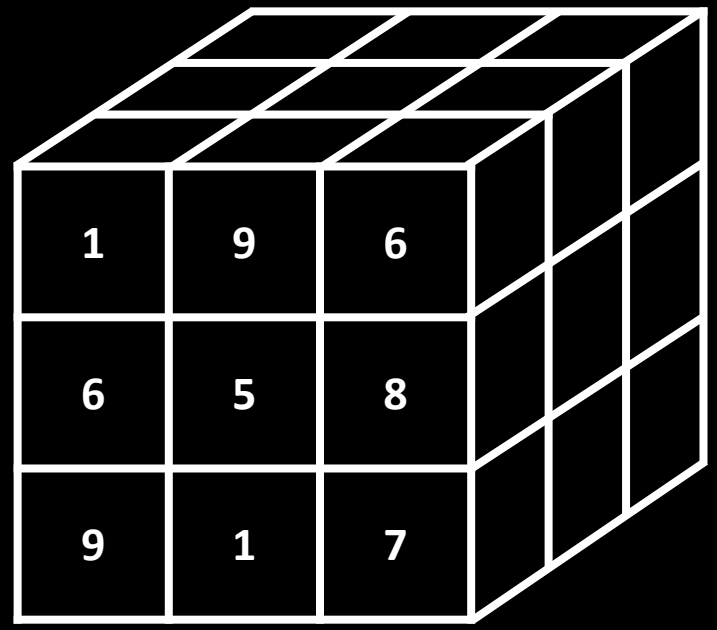
- Array is a fundamental and linear data structure which is a collection of same type elements that are stored at contiguous memory locations where the elements are identified / accessed by their indexes.



1D Array

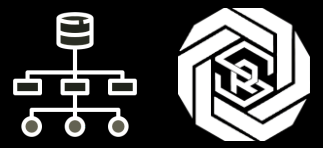


2D Array



3D Array

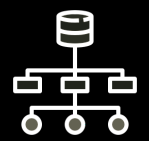
Real-World Analogy



Row of Lockers

- Imagine a **row of school lockers**.
- Each locker is identified by a **number** (**index**).
- Each locker stores **one item** (**element**).
- All lockers are **next to each other** (**contiguous memory**).
- You can directly open any locker using its number without checking others — just like accessing an array element using its index.

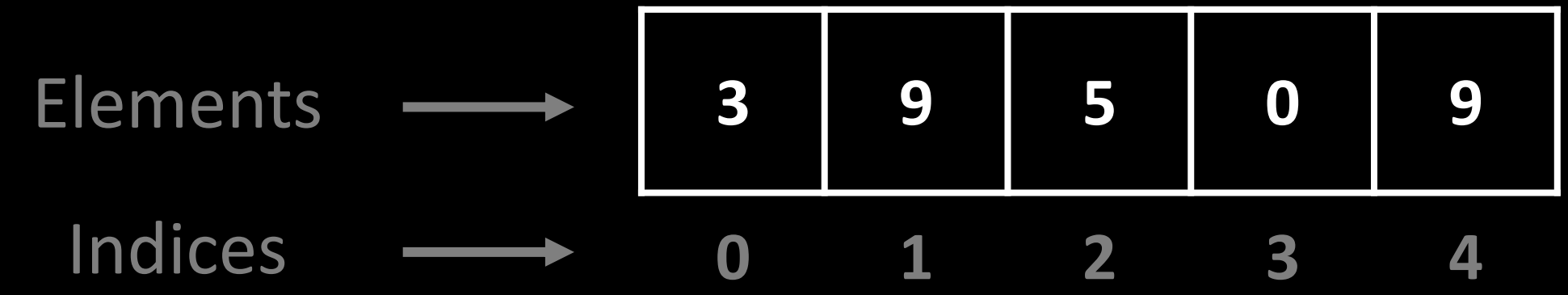
Types



1. One-Dimensional Array: An Array represented as a row, where elements are stored one after the other.
2. Multi-Dimensional Array: An array with more than one dimension. It is used to store complex data in the form of tables.
   - Two-Dimensional Array: An Array which can be considered as an array of arrays or as a matrix containing rows and columns.
   - Three-Dimensional Array: An Array which can be considered an array of two-dimensions containing three dimensions.
   - And so on…

Types

One-Dimensional Array



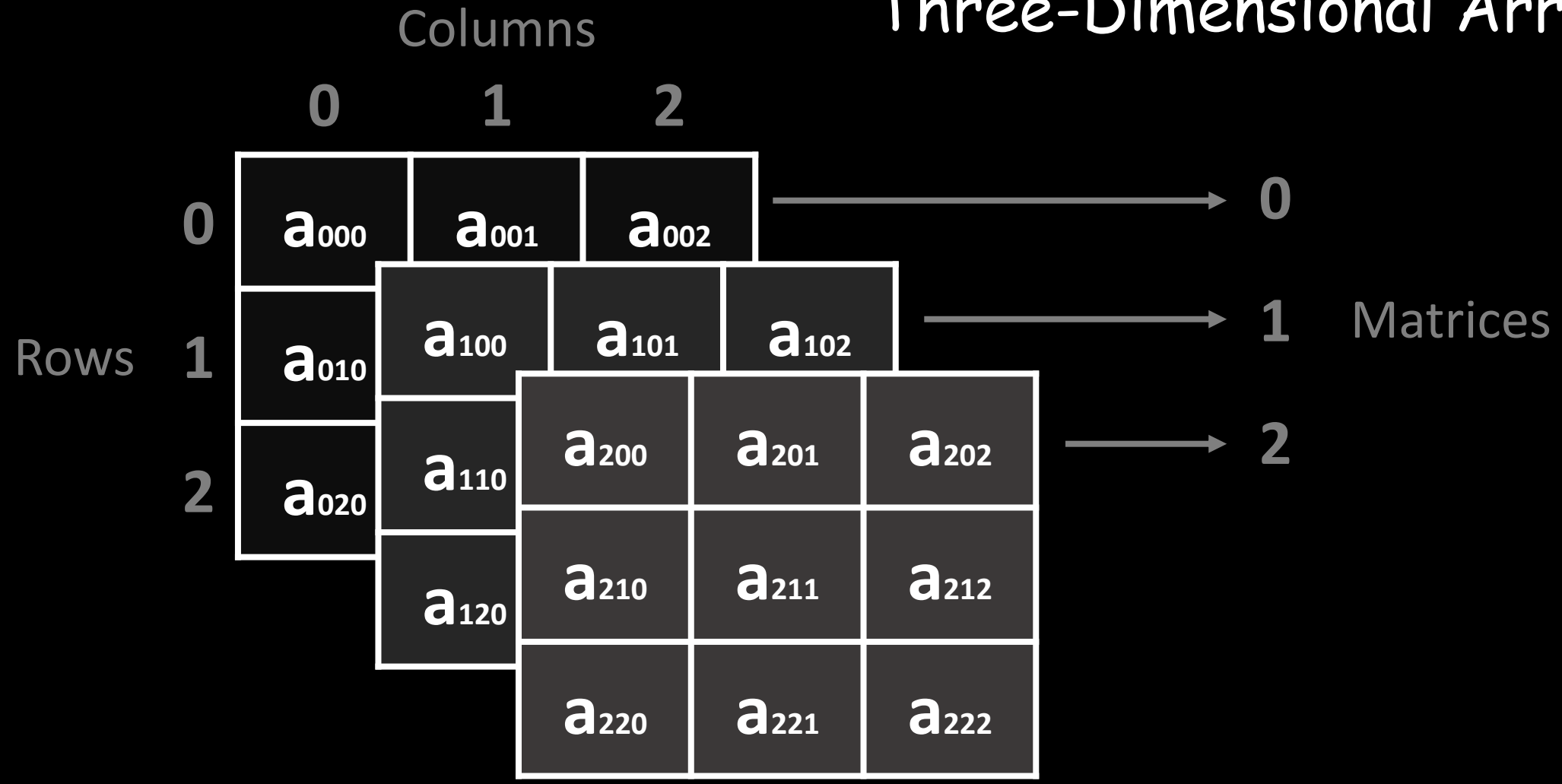
Types

Two-Dimensional Array

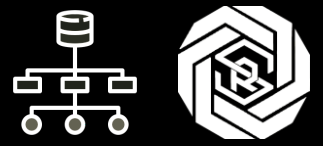
Rows \ Columns	0	1	2	3
0	3 a_{00}	9 a_{01}	0 a_{02}	2 a_{03}
1	9 a_{10}	5 a_{11}	6 a_{12}	1 a_{13}

Types

Three-Dimensional Array



Syntax

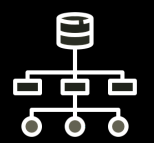


```
dataType[] arrayName = new dataType[size]; // 1D

dataType[][] arrayName = new dataType[size1][size2]; // 2D

dataType[][][] arrayName = new dataType[size1][size2][size3]; // 3D
```

Initialization of 1D-Array

```
int arrayName[] = {1, 2, 3}; // or int[] arrayName = {1, 2, 3};

char arrayName[] = {'a', 'b', 'c'};

float arrayName[] = {1.0f, 2.0f, 3.0f};
```

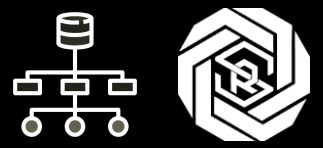
Features / Characteristics



- **Random Access** - Elements can be accessed in a random fashion.
- Index always starts from **0**.
- **Homogeneous Data**: All elements are of the same type.
- **Indexed Access**: Each element can be accessed directly using an index.
- **Fixed Size**: Once created, the size of an array cannot be changed.
- **Contiguous Memory**: Elements are stored next to each other in memory for fast access.
- **Default Values**: Arrays are initialized with default values depending on the array type and the programming language. In Java:

Data Type	byte	short	int	long	float	double	char	boolean
Default Value	0	0	0	0L	0.0f	0.0d	\u0000 (null)	false

Operations



- update(): Used to update an element in an array.
- display() (or Traversal): Used to traverse through all the elements of an array and display them.
- get(): Used to retrieve a value using an index.

Example

Step 1: Create an Array

int[] arr = new int[5];

An integer array arr of size 5 is created.

(Currently all elements are 0 by default)

0	0	0	0	0

Step 2: Insert Elements

arr[0] = 10; arr[1] = 20; arr[2] = 80;

arr[3] = 40; arr[4] = 50;

Now the array contains:

10	20	80	40	50

Step 3: Update an Element

arr[2] = 30; // updating index 2

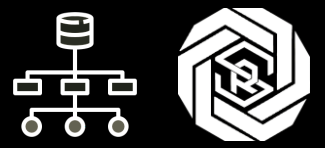
Now the array becomes:

10	20	30	40	50

Step 4: Retrieve an Element

arr[3] gives you 40

Algorithm



For One-Dimensional Array

Step 1: Start

Step 2: Input the size of the array.

Step 3: Create an integer array of given size.

Step 4: For each index i from 0 to size - 1:

Input the element and store it in array[i].

Step 5: To Update an element:

If index is valid → array[index] = value
Else → Display "Index out of bounds."

Step 6: To Get an element:

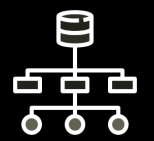
If index is valid → Display array[index]
Else → Display "Index out of bounds."

Step 7: To Display all elements:

Print each element in array.

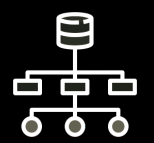
Step 8: Stop

Implementation



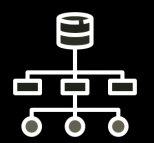
```java
import java.util.Scanner;

public class OneDimensionalArray {
  private int[] array;

  // Constructor to initialize the array
  public OneDimensionalArray(int size) {
    array = new int[size];
    Scanner scanner = new Scanner(System.in);
    System.out.println("\nEnter elements");

    for (int i = 0; i < size; i++) {
      System.out.print("Index " + i + ": ");
      array[i] = scanner.nextInt();
    }
  }

  // Method to update an element at a specified index
  public void update(int index, int value) {
    if (index >= 0 && index < array.length) {
      array[index] = value;
    } else {
      System.out.println("Index out of bounds.");
    }
  }

  // Method to retrieve a value at a specified index
  public void get(int index) {
    System.out.println();
    if (index >= 0 && index < array.length) {
      System.out.println("Value at index " + index + ": "
+ array[index]);
    } else {
      System.out.println("Index out of bounds.");
    }
  }

  // Method to display all elements of the array
  public void display() {
    System.out.println();
    for (int element : array) {
      System.out.print(element + " ");
    }
    System.out.println();
  }

  public static void main(String[] args) {
    int op, index, value;
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter the size of the array: ");
    int size = scanner.nextInt();
    OneDimensionalArray myArray = new
OneDimensionalArray(size);

    do {
      System.out.println("\nOperations");
      System.out.println("1. Update");
      System.out.println("2. Get");
      System.out.println("3. Display");
      System.out.println("4. Exit");
      System.out.print("Select an operation: ");
      op = scanner.nextInt();

      switch (op) {
        case 1:
          System.out.print("\nEnter Index: ");
          index = scanner.nextInt();
          System.out.print("Enter Value: ");
          value = scanner.nextInt();
          myArray.update(index, value);
          break;
        case 2:
          System.out.print("\nEnter Index: ");
          index = scanner.nextInt();
          myArray.get(index);
          break;
        case 3:
          myArray.display();
          break;
        case 4:
          System.out.println("Exiting the program.");
          System.exit(0);
          break;
        default:
          System.out.println("\nPlease enter a valid
option.");
      }
    } while (op != 4);
    scanner.close();
  }
}
```

Use Cases



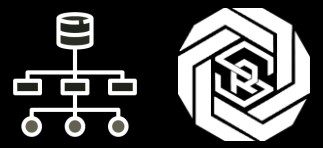
- When You Know the Number of Elements in Advance.
- When You Need Fast Access (Random Access).
- When Data is Homogeneous.
- When Memory Efficiency is Important.
- When Frequent Access is Needed, But Not Frequent Insertions or Deletions.
- When You Need to Perform Numerical Computations.
- When Order Matters.

Applications



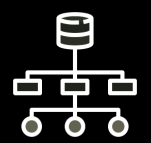
- Implementation of static data structures such as stacks, queues, Hash Tables, Linked Lists, Array Lists, Heaps, Vectors, Matrices, Trees, Graphs, etc.
- Database records are usually implemented as arrays.
- Used in lookup tables by the computer.
- Sorting and Searching algorithms (they heavily use arrays).
- Machine learning datasets (stored as arrays/matrices).
- Network packets / buffers use arrays internally.

Limitations



- Have a fixed size(Static array).
- Cannot store Heterogeneous elements.
- Cannot be used where we have operations like insert in middle, delete from middle or search in unsorted data.

Complexities



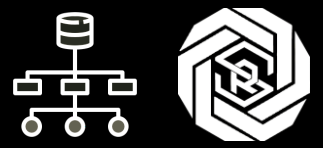
- Time Complexity

Operation	Best Case	Average Case	Worst Case
Insertion	Ω(N)	Θ(N)	O(N)
Update	Ω(1)	Θ(1)	O(1)
Get	Ω(1)	Θ(1)	O(1)
Traversal	Ω(N)	Θ(N)	O(N)
Searching	Ω(1)	Θ(N)	O(N)

- Space Complexity

O(N) - Space required to store the array elements.

Exceptions



- ArrayIndexOutOfBoundsException

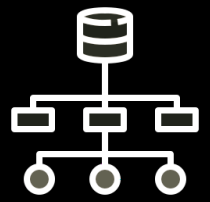
```
int[] arr = new int[3];
System.out.println(arr[5]); // ArrayIndexOutOfBoundsException
```

- NullPointerException

```
arr = null;
System.out.println(arr.length); // NullPointerException
```

- ArrayStoreException

```
Object[] o = new String[3];
o[0] = 10; // ArrayStoreException
```



End of Notes