🧩 Java Arrays – Complete Notes

---

1. Array Introduction

An array in Java is a data structure that stores multiple elements of the **same data type** in a **contiguous memory location**.
It is used to store a fixed-size sequence of elements like integers, strings, or objects.

Example:

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

Key Idea:
Instead of creating multiple variables (int a, b, c;), we can use one array to store them all.

---

2. Definition of Array

An array is a **collection of similar type elements**, stored under one variable name, and accessed using an **index**.

In simple terms:

"An array is a container that holds multiple values of the same type."

---

3. Need of Array (Why Arrays?)

Without arrays, we need multiple variables for multiple data.
Arrays make it easy to:

- Store large data in one place
- Access elements using index
- Perform operations using loops

Example:

```
int marks[] = new int[5];  // instead of int mark1, mark2, mark3, ...
```

---

4. Array Declaration

Declaring an array means telling Java the **type of data** and **how many elements** it will store.

Syntax:

```
dataType[] arrayName;
```

Example:

```
int[] arr;     // Recommended

int arr[];     // Also valid
```

---

5. Array Initialization

Assigning values to the array elements.

Types:

1. **Static Initialization**
2. int[] arr = {1, 2, 3, 4};
3. **Dynamic Initialization**
4. int[] arr = new int[5];
5. arr[0] = 10;
6. arr[1] = 20;

---

6. Types of Array

a) Single Dimensional Array

A simple list of elements.

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

b) Multi-Dimensional Array

Array of arrays (like matrix).

int[][] matrix = { {1,2}, {3,4}, {5,6} };

c) Jagged Array

An array with rows of **different lengths**.

int[][] jagged = { {1,2,3}, {4,5}, {6} };

---

7. Array Memory Representation

- Stored in **heap memory** (since arrays are objects in Java).
- Index starts from **0**.
- Each element stored **contiguously**.

---

8. Default Values in Array

If not initialized, array elements get **default values**:

Data Type	Default Value
int, byte, short, long	0
float, double	0.0
char	'\u0000' (null char)
boolean	false
Object	null

---

9. Accessing Array Elements

Access elements using **index**:

```
System.out.println(arr[0]); // first element
arr[2] = 99;                // change value
```

---

10. Traversing Array

To visit each element of the array.

Using for loop

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

Using for-each loop

```
for(int x : arr)
   System.out.println(x);
```

Using while loop

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

---

11. Array Input from User

```
Scanner sc = new Scanner(System.in);
int[] arr = new int[5];
for(int i=0; i<5; i++) {
   arr[i] = sc.nextInt();
```

}

---

12. Length Property in Array

Used to get the number of elements in an array.

System.out.println(arr.length);

---

13. Array Index & Out of Bound Exception

If you access an index that doesn't exist →
ArrayIndexOutOfBoundsException

Example:

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

System.out.println(arr[3]); // Error!

---

14. Operations on Array

a) Insertion

arr[2] = 50;

b) Deletion

Can't directly delete – you create a new array without that element.

c) Searching

Find element by value.

d) Sorting

Arrange elements in order.

e) Updating

Change the value at a given index.

---

15. Common Algorithms using Arrays

a) Linear Searc

b) Binary Search

c) Bubble Sort

d) Selection Sort

e) Insertion Sort

these topic covered on dsa note.

---

16. Array with Methods (Passing Array to Methods)

You can pass an array as a parameter.

```
void printArray(int[] arr){
  for(int x : arr)
      System.out.println(x);
}
```

---

17. Returning Array from Method

A method can return an array.

```
int[] getArray(){
  return new int[]{1,2,3,4};
}
```

---

18. Anonymous Arrays

Arrays without a name.

```
printArray(new int[]{10,20,30});
```

---

19. Copying Arrays

a) Manual Copy

```
for(int i=0; i<arr.length; i++)
  newArr[i] = arr[i];
```

b) System.arraycopy()

```
System.arraycopy(arr, 0, newArr, 0, arr.length);
```

c) Arrays.copyOf()

```
int[] newArr = Arrays.copyOf(arr, arr.length);
```

d) clone()

```
int[] newArr = arr.clone();
```

---

21. For-each Loop with Array

Simpler way to access array elements:

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

Limitation: Cannot modify array elements directly.

---

22. Multidimensional Array Operations

Traversal:

```
for(int i=0; i<mat.length; i++)
  for(int j=0; j<mat[i].length; j++)
    System.out.print(mat[i][j] + " ");
```

Input:

```
for(int i=0; i<2; i++)
  for(int j=0; j<3; j++)
    mat[i][j] = sc.nextInt();
```

---

23. Jagged Array Concept & Example

Rows have different column lengths.

```
int[][] jagged = new int[3][];
jagged[0] = new int[3];
jagged[1] = new int[2];
jagged[2] = new int[4];
```

---

24. Array vs ArrayList

Feature	**Array**	**ArrayList**
Size	Fixed	Dynamic
Type	Primitive & Objects	Objects only
Performance	Faster	Slightly slower
Length	arr.length	list.size()
Package	java.lang	java.util

---

25. Advantages of Arrays

- Easy to access using index
- Memory-efficient (contiguous)
- Easy to traverse and manipulate
- Supports random access

---

26. Limitations of Arrays

- Fixed size (cannot grow/shrink)
- Difficult insertion/deletion
- Can store only similar data type
- No built-in methods for dynamic resizing

---

27. Real-life Examples of Arrays in Java

- Storing marks of students
- Keeping daily temperature records
- Managing stock prices
- Game leaderboards
- Storing employee IDs, salaries, etc.