Java - Arrays

1. Introduction to Arrays

- An **array** is a collection of elements of the **same data type**, stored in **contiguous memory locations**.
- Arrays allow **efficient storage and retrieval** of data using **index values**.
- They are **fixed in size** (once created, size cannot be changed).

Example:

```
int[] numbers = new int[5];  // stores 5 integers
```

---

2. Characteristics of Arrays

- Homogeneous elements (all elements must be of the same type).
- Indexed starting from **0** to **n-1**.
- Stored in contiguous memory.
- Fixed size (cannot grow/shrink dynamically).
- Direct access using index.

---

3. Array Declaration and Initialization

Type 1: Declaration then Memory Allocation

```
int[] marks;          // Declaration
marks = new int[5];   // Memory allocation
```

- First declares the array variable, then allocates memory separately.

Type 2: Declaration + Memory Allocation Together

```
int[] marks = new int[5]; // Declaration + Memory Allocation
```

- Combines declaration and memory allocation in a single line.
- All elements are initialized to their **default values** (`0` for int, `null` for objects, `false` for boolean, etc.).

Type 3: Declaration + Initialization with Values

```
int[] marks = {1, 2, 3, 4, 5, 6};  // Declaration + Initialization
```

- Array is created and initialized with given values.
- The size of the array is determined automatically.

4. Accessing Array Elements

- Elements are accessed using **index numbers**.
- Index starts at **0** and ends at `length - 1`.

Example:

```
int[] arr = {10, 20, 30, 40, 50};
System.out.println(arr[0]);  // Output: 10
System.out.println(arr[4]);  // Output: 50
```

⇒ If we try to access an index outside the range → `ArrayIndexOutOfBoundsException`.

5. Default Values in Arrays

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

6. Types of Arrays

1. One-Dimensional Array

- A linear collection of elements.

Example:

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

2. Multi-Dimensional Array (2D, 3D, ...)

- Arrays of arrays.
- Mostly used for matrix-like data.

Example (2D array):

```
int[][] matrix = new int[3][3]; // 3×3 matrix
int[][] predefined = {{1,2,3},{4,5,6},{7,8,9}};
```

3. Jagged Array

- Array of arrays with **different column sizes**.

Example:

```
int[][] jagged = new int[3][];
jagged[0] = new int[2];  // row 1 has 2 elements
jagged[1] = new int[3];  // row 2 has 3 elements
jagged[2] = new int[1];  // row 3 has 1 element
```

7. Array Operations

Traversing an Array

1. **For Loop**

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

2. **Enhanced For Loop (for-each)**

   ```
   for(int element : arr) {
       System.out.println(element);
   }
   ```

3. **Using While Loop**

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

Copying Arrays

```
int[] original = {1,2,3};
int[] copy = Arrays.copyOf(original, original.length);
```

Sorting Arrays

```
Arrays.sort(arr);
```

Searching Arrays

```
int index = Arrays.binarySearch(arr, 30);
```

8. Advantages of Arrays

- Easy and fast data access via indexing.
- Memory efficient for fixed-size data.
- Better performance compared to collections in simple cases.

9. Disadvantages of Arrays

- Fixed size (cannot resize after creation).
- Insertion/deletion is costly (elements must be shifted).
- Cannot store heterogeneous data.
- No built-in methods for dynamic operations (unlike ArrayList).

10. Common Errors with Arrays

1. **ArrayIndexOutOfBoundsException**

   Occurs when accessing index outside range.
2. **NullPointerException**

If array is declared but not allocated memory.

3. **Initialization mistakes**

   Forgetting default values and assuming garbage values.

---

11. Arrays Utility Class (`java.util.Arrays`)

Some useful methods:

- `Arrays.sort(arr)`
- `Arrays.toString(arr)`
- `Arrays.equals(arr1, arr2)`
- `Arrays.fill(arr, value)`
- `Arrays.copyOf(arr, newLength)`
- `Arrays.binarySearch(arr, key)`

---

12. Real-Life Examples of Arrays

- Storing marks of students.
- Matrix representations in games.
- Image pixel storage.
- Database records caching.

🔑 Key Takeaways

- Arrays are fixed-size, contiguous memory structures.
- Provide fast random access using indexes.
- Can be 1D, multi-D, or jagged.
- Always initialized with default values.
- For dynamic size → prefer `ArrayList`.