



1D and 2D Arrays in java

In Java, **arrays** are used to store multiple values of the same type in a single variable. They can be **1D (one-dimensional)** or **2D (two-dimensional)**. Here's a clear explanation of both:

♦ 1D Array in Java

✓ Definition:

A 1D array is a **single row** of elements. It stores a list of items in a **linear** fashion.

✓ Declaration and Initialization:

```
java

int[] numbers = new int[5]; // Declaration with size
numbers[0] = 10;           // Assigning values

// Or directly initialize
int[] numbers2 = {10, 20, 30, 40, 50};
```

✓ Accessing Elements:

```
java

System.out.println(numbers2[2]); // Output: 30
```

✓ Traversing 1D Array:

```
java

for (int i = 0; i < numbers2.length; i++) {
    System.out.println(numbers2[i]);
}
```

✓ Enhanced for loop:

```
java

for (int num : numbers2) {
    System.out.println(num);
}
```

♦ 2D Array in Java

✓ Definition:

A 2D array is like a **matrix** or **table** with rows and columns.

✓ Declaration and Initialization:

```
java

int[][] matrix = new int[2][3]; // 2 rows, 3 columns
matrix[0][0] = 1;
matrix[0][1] = 2;
matrix[1][2] = 6;

// Or directly initialize
int[][] matrix2 = {
    {1, 2, 3},
    {4, 5, 6}
};
```

✓ Accessing Elements:

```
java

System.out.println(matrix2[1][2]); // Output: 6
```

✓ Traversing 2D Array:

```
java

for (int i = 0; i < matrix2.length; i++) {
    for (int j = 0; j < matrix2[i].length; j++) {
        System.out.print(matrix2[i][j] + " ");
    }
    System.out.println();
}
```

◆ Summary Table

Feature	1D Array	2D Array
Shape	Linear (like a list)	Matrix (rows × columns)
Syntax	<code>int[] arr = new int[5];</code>	<code>int[][] arr = new int[2][3];</code>
Access element	<code>arr[2]</code>	<code>arr[1][2]</code>
Traversal	Single loop	Nested loop

Would you like examples using **strings**, **user input**, or practical use cases like **searching/sorting** in arrays?



yes

Great! Let's go through **examples of 1D and 2D arrays** using:

1. ♦ **Strings**
2. ♦ **User Input**
3. ♦ **Search**
4. ♦ **Sort (1D only)**

♦ 1. 1D Array with Strings

✓ Example: Store and print names

```
java

public class StringArrayExample {
    public static void main(String[] args) {
        String[] names = {"Ram", "Shyam", "Geeta", "Sita"};

        for (String name : names) {
            System.out.println(name);
        }
    }
}
```

♦ 2. 1D Array – User Input Example

```
java

import java.util.Scanner;

public class UserInput1D {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int[] numbers = new int[5];

        System.out.println("Enter 5 numbers:");
        for (int i = 0; i < numbers.length; i++) {
            numbers[i] = sc.nextInt();
        }

        System.out.println("You entered:");
        for (int num : numbers) {
            System.out.println(num);
        }
    }
}
```

♦ 3. 1D Array – Search Element

```
java

public class SearchArray {
    public static void main(String[] args) {
        int[] arr = {10, 20, 30, 40, 50};
        int search = 30;
        boolean found = false;

        for (int num : arr) {
```

```

        if (num == search) {
            found = true;
            break;
        }
    }

    if (found)
        System.out.println(search + " found in array.");
    else
        System.out.println(search + " not found.");
}
}

```

◆ 4. 1D Array – Sort Elements (Ascending)

```

java

import java.util.Arrays;

public class SortArray {
    public static void main(String[] args) {
        int[] arr = {50, 20, 40, 10, 30};

        Arrays.sort(arr); // Built-in sort

        System.out.println("Sorted array:");
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }
}

```

◆ 5. 2D Array with Strings

```

java

public class String2DArray {
    public static void main(String[] args) {
        String[][] cities = {
            {"Delhi", "Mumbai"},
            {"Hyderabad", "Chennai"}
        };

        for (int i = 0; i < cities.length; i++) {
            for (int j = 0; j < cities[i].length; j++) {
                System.out.println("City[" + i + "][" + j + "] = " + cities[i][j]);
            }
        }
    }
}

```

◆ 6. 2D Array – User Input Example

```

java

import java.util.Scanner;

public class UserInput2D {
    public static void main(String[] args) {




```

```
Scanner sc = new Scanner(System.in);
int[][] matrix = new int[2][2];

System.out.println("Enter 4 numbers (2x2 matrix):");
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
        matrix[i][j] = sc.nextInt();
    }
}

System.out.println("Matrix is:");
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
        System.out.print(matrix[i][j] + " ");
    }
    System.out.println();
}
}
```

Would you like to go deeper into:

-  Matrix operations (like transpose, addition, multiplication)?
-  ArrayList (dynamic array)?
-  Jagged Arrays (2D arrays with unequal rows)?

Let me know your next area of interest!



Array methods

In Java, **arrays** are **objects**, but they don't have many built-in methods like **ArrayList**. However, Java provides useful **utility methods** for arrays via the **java.util.Arrays** class.

◆ Commonly Used Array Methods in Java

◆ 1. Arrays.toString() – Print array as string

```
java

import java.util.Arrays;

int[] arr = {10, 20, 30};
System.out.println(Arrays.toString(arr)); // Output: [10, 20, 30]
```

◆ 2. Arrays.sort() – Sort elements

```
java

int[] arr = {40, 10, 30, 20};
Arrays.sort(arr);
System.out.println(Arrays.toString(arr)); // Output: [10, 20, 30, 40]
```

♦ 3. `Arrays.equals()` – Compare two arrays (element by element)

```
java

int[] arr1 = {1, 2, 3};
int[] arr2 = {1, 2, 3};
System.out.println(Arrays.equals(arr1, arr2)); // Output: true
```

♦ 4. `Arrays.copyOf()` – Copy array with new size

```
java

int[] original = {1, 2, 3};
int[] copy = Arrays.copyOf(original, 5);
System.out.println(Arrays.toString(copy)); // Output: [1, 2, 3, 0, 0]
```

♦ 5. `Arrays.copyOfRange()` – Copy specific range

```
java

int[] arr = {10, 20, 30, 40, 50};
int[] range = Arrays.copyOfRange(arr, 1, 4); // from index 1 to 3
System.out.println(Arrays.toString(range)); // Output: [20, 30, 40]
```

♦ 6. `Arrays.fill()` – Fill array with a value

```
java

int[] arr = new int[5];
Arrays.fill(arr, 100);
System.out.println(Arrays.toString(arr)); // Output: [100, 100, 100, 100, 100]
```

♦ 7. `Arrays.binarySearch()` – Search in sorted array

```
java

int[] arr = {10, 20, 30, 40, 50};
int index = Arrays.binarySearch(arr, 30);
System.out.println("Found at index: " + index); // Output: 2
```

⚠ Array must be **sorted** before using `binarySearch()`

♦ 8. `Arrays.deepToString()` – For multidimensional arrays

```
java

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

```
System.out.println(Arrays.deepToString(matrix)); // Output: [[1, 2], [3, 4]]
```

◆ Summary Table

Method	Description
<code>toString()</code>	Prints 1D array as a string
<code>sort()</code>	Sorts the array in ascending order
<code>equals()</code>	Compares two arrays
<code>copyOf()</code>	Copies the array to new array
<code>copyOfRange()</code>	Copies a part of the array
<code>fill()</code>	Fills the array with a single value
<code>binarySearch()</code>	Finds index of element (sorted only)
<code>deepToString()</code>	Prints multi-dimensional arrays

Would you like me to explain **ArrayList methods** (dynamic array) next? Or help with **practicing questions** on arrays (like reverse, find max/min, etc.)?



help with practicing questions on arrays (like reverse, find max/min, etc.)?

Perfect! Practicing array problems is a great way to build strong programming logic.

◆ Array Practice Questions (with Code)

✓ 1. Reverse an Array

```
java

public class ReverseArray {
    public static void main(String[] args) {
        int[] arr = {10, 20, 30, 40, 50};

        System.out.print("Reversed Array: ");
        for (int i = arr.length - 1; i >= 0; i--) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

✓ 2. Find Maximum and Minimum Element

```
java

public class MaxMinArray {
    public static void main(String[] args) {
        int[] arr = {25, 10, 55, 5, 40};
```

```

    int max = arr[0];
    int min = arr[0];

    for (int num : arr) {
        if (num > max) max = num;
        if (num < min) min = num;
    }

    System.out.println("Max: " + max);
    System.out.println("Min: " + min);
}
}

```

✓ 3. Sum and Average of Elements

```

java

public class SumAverage {
    public static void main(String[] args) {
        int[] arr = {10, 20, 30, 40, 50};
        int sum = 0;

        for (int num : arr) {
            sum += num;
        }

        double avg = (double) sum / arr.length;

        System.out.println("Sum = " + sum);
        System.out.println("Average = " + avg);
    }
}

```

✓ 4. Count Even and Odd Numbers

```

java

public class CountEvenOdd {
    public static void main(String[] args) {
        int[] arr = {11, 22, 33, 44, 55};
        int even = 0, odd = 0;

        for (int num : arr) {
            if (num % 2 == 0) even++;
            else odd++;
        }

        System.out.println("Even Count: " + even);
        System.out.println("Odd Count: " + odd);
    }
}

```

✓ 5. Check if Array is Palindrome (e.g., {1, 2, 3, 2, 1})

```

java

public class PalindromeArray {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 2, 1};
        boolean isPalindrome = true;
    }
}

```



```

        for (int i = 0; i < arr.length / 2; i++) {
            if (arr[i] != arr[arr.length - 1 - i]) {
                isPalindrome = false;
                break;
            }
        }

        System.out.println("Is Palindrome? " + isPalindrome);
    }
}

```

✓ 6. Remove Duplicates from Array (Simple version)

```

java

import java.util.HashSet;

public class RemoveDuplicates {
    public static void main(String[] args) {
        int[] arr = {10, 20, 10, 30, 40, 20};
        HashSet<Integer> set = new HashSet<>();

        System.out.println("After removing duplicates:");
        for (int num : arr) {
            if (set.add(num)) {
                System.out.print(num + " ");
            }
        }
    }
}

```

✓ 7. Find Second Largest Element

```

java

public class SecondLargest {
    public static void main(String[] args) {
        int[] arr = {12, 35, 1, 10, 34, 1};

        int first = Integer.MIN_VALUE;
        int second = Integer.MIN_VALUE;

        for (int num : arr) {
            if (num > first) {
                second = first;
                first = num;
            } else if (num > second && num != first) {
                second = num;
            }
        }

        System.out.println("Second largest: " + second);
    }
}

```

Want More?

Would you like:

- Practice questions on **2D arrays** (like matrix addition, transpose)?
- Convert these into **methods**?
- Assignments or mini-projects based on arrays?

Let me know and I'll tailor it for you.



String and StringBuffer, StringBuilder

Great topic! Let's explore the differences and usage of **String**, **StringBuffer**, and **StringBuilder** in Java.

♦ 1. String in Java

✓ Key Points:

- **Immutable**: Once created, it cannot be changed.
- Every modification creates a **new object** in memory.
- Stored in the **String pool**.

✓ Example:

```
java

String s = "Hello";
s = s.concat(" World");
System.out.println(s); // Output: Hello World
```

A new object is created for "Hello World", the original "Hello" remains unchanged in memory.

♦ 2. StringBuffer in Java

✓ Key Points:

- **Mutable**: Can be modified after creation.
- **Thread-safe** (synchronized) – safe for multi-threading.
- Slightly slower than **StringBuilder**.

✓ Example:

```
java

StringBuffer sb = new StringBuffer("Hello");
sb.append(" World");
System.out.println(sb); // Output: Hello World
```

◆ 3. StringBuilder in Java

✓ Key Points:

- **Mutable** (like `StringBuffer`)
- **Not thread-safe** (not synchronized)
- **Faster** than `StringBuffer` in single-threaded scenarios

✓ Example:

```
java

StringBuilder sb = new StringBuilder("Hello");
sb.append(" World");
System.out.println(sb); // Output: Hello World
```

◆ Differences Summary

Feature	String	StringBuffer	StringBuilder
Mutability	Immutable	Mutable	Mutable
Thread-safe	Yes (because immutable)	Yes (synchronized)	No (not synchronized)
Performance	Slow (creates new object)	Slower (due to synchronization)	Fastest
Use case	Fixed strings	Multi-threaded apps	Single-threaded apps

◆ Example Showing Difference:

```
java

public class StringTypes {
    public static void main(String[] args) {
        String s = "Java";
        s.concat(" Programming");
        System.out.println("String: " + s); // Output: Java

        StringBuffer sb = new StringBuffer("Java");
        sb.append(" Programming");
        System.out.println("StringBuffer: " + sb); // Output: Java Programming

        StringBuilder sbl = new StringBuilder("Java");
        sbl.append(" Programming");
        System.out.println("StringBuilder: " + sbl); // Output: Java Programming
    }
}
```

◆ When to Use What?

Scenario	Use
Read-only or constant strings	String

Scenario	Use
Need mutable string with thread-safety	StringBuffer
Need mutable string, better performance	StringBuilder

Would you like:

- Common **methods** used in each class?
- Practice programs or interview questions based on this?

Let me know!



yes

Excellent! Let's now explore:

✓ Common Methods in String, StringBuffer, and StringBuilder

♦ 1. Common String Methods

java

```
String str = "Java Programming";
```

Method	Description	Example	Output
length()	Returns length	str.length()	16
charAt(int)	Character at index	str.charAt(5)	P
substring(int, int)	Extract part of string	str.substring(5, 16)	Programming
toLowerCase()	Convert to lowercase	str.toLowerCase()	java programming
toUpperCase()	Convert to uppercase	str.toUpperCase()	JAVA PROGRAMMING
equals()	Compare content	str.equals("Java")	false
equalsIgnoreCase()	Ignore case compare	"JAVA".equalsIgnoreCase("java")	true
contains()	Check substring	str.contains("Pro")	true
replace()	Replace character or string	str.replace("a", "@")	J@v@ Progr@mming
split()	Split string	str.split(" ")	["Java", "Programming"]

Method	Description	Example	Output
<code>trim()</code>	Remove whitespace	<code>" Java ".trim()</code>	<code>"Java"</code>

◆ 2. Common StringBuffer Methods

```
java
StringBuffer sb = new StringBuffer("Hello");
```

Method	Description	Example	Output
<code>append()</code>	Adds text	<code>sb.append(" World")</code>	<code>Hello World</code>
<code>insert(int, str)</code>	Insert text	<code>sb.insert(5, ",")</code>	<code>Hello, World</code>
<code>replace(start, end, str)</code>	Replace part	<code>sb.replace(0, 5, "Hi")</code>	<code>Hi, World</code>
<code>delete(start, end)</code>	Delete part	<code>sb.delete(2, 5)</code>	<code>HiWorld</code>
<code>reverse()</code>	Reverses string	<code>sb.reverse()</code>	<code>dlroW ,iH</code>
<code>capacity()</code>	Current capacity	<code>sb.capacity()</code>	<code>Initial = 16 + length</code>
<code>charAt()</code>	Get char at index	<code>sb.charAt(1)</code>	<code>'i'</code>

◆ 3. Common StringBuilder Methods

```
java
StringBuilder sbl = new StringBuilder("Welcome");
```

Same as `StringBuffer` (but not thread-safe). Example methods:

Method	Description	Example
<code>append()</code>	<code>sbl.append(" to Java")</code> → <code>Welcome to Java</code>	
<code>delete()</code>	<code>sbl.delete(0, 3)</code> → removes <code>"Wel"</code>	
<code>insert()</code>	<code>sbl.insert(3, "X")</code> → <code>WelXcome</code>	
<code>reverse()</code>	<code>sbl.reverse()</code> → <code>emocleW</code>	

◆ Practice Programs

✓ 1. Count Vowels in a String

```
java
public class CountVowels {
    public static void main(String[] args) {
        String str = "Hello Java";
        int count = 0;
        for (char c : str.toLowerCase().toCharArray()) {
            if ("aeiou".indexOf(c) != -1) {
```

```

        count++;
    }
}
System.out.println("Vowel count: " + count);
}
}

```

✓ 2. Reverse a String using StringBuilder

```

java

public class ReverseString {
    public static void main(String[] args) {
        String str = "Automation";
        StringBuilder sb = new StringBuilder(str);
        System.out.println("Reversed: " + sb.reverse());
    }
}

```

✓ 3. Check if Two Strings are Anagrams

```

java

import java.util.Arrays;

public class AnagramCheck {
    public static void main(String[] args) {
        String a = "listen";
        String b = "silent";

        char[] arr1 = a.toCharArray();
        char[] arr2 = b.toCharArray();

        Arrays.sort(arr1);
        Arrays.sort(arr2);

        boolean result = Arrays.equals(arr1, arr2);
        System.out.println("Are Anagrams? " + result);
    }
}

```

◆ Interview Questions

Question	Quick Answer
What is the difference between String, StringBuffer, and StringBuilder?	Immutable vs mutable, thread-safety
Why is String immutable in Java?	For security, caching, thread-safety, string pool optimization
Which is faster: StringBuffer or StringBuilder?	StringBuilder is faster (no synchronization)
Can StringBuilder be used in multithreading?	Not recommended (not thread-safe)

Would you like a **mini-project using String operations**, or **MCQs and coding tasks** for interview prep?