Coding Age

Java : Array

What is an Array?

-An **array** in Java is a collection of similar data types stored in contiguous memory locations. Arrays can hold multiple values of the same data type and provide a way to access those values using an index.

1D Arrays in Java

Key Features of Arrays:

- **Homogeneous**: All elements in an array must be of the same type (e.g., all integers).
- **Fixed Size**: Once an array is created, its size cannot be changed.
- **Zero-Indexed**: The first element is accessed with index `0`, the second with index `1`, and so on.

How to Declare an Array?

To declare an array in Java, you specify the data type followed by square brackets.

Syntax:

```
dataType[] arrayName;
// or
dataType arrayName[]; // Valid but less commonly used
```

Example:

```
int[] arr; // Declares an array of integers
```

How to Create (Allocate) an Array?

After declaring an array, you need to allocate memory for it using the new keyword.

Example:

```
arr = new int[5]; //Creates an array to hold 5 integers
```

- You can also declare and create the array in one step:

```
int[] arr = new int[5];
// Declares and creates an array of size 5
```

How to Initialize an Array?

- You can initialize an array while declaring it:

Example:

```
int[] arr = {4, 2, 3};
// Initializes the array with values
```

- Or you can create an array and initialize it separately:

```
int[] arr = new int[3]; // Creates an array of size 3
arr[0] = 4; // Assigns value 4 to the first element
arr[1] = 2; // Assigns value 2 to the second element
arr[2] = 3; // Assigns value 3 to the third element
```

How to Find the Size of an Array?

In Java, you can find the size of an array using the `.length` property.

Example:

```
int[] arr = {5, 6, 2};
int size = arr.length; // size will be 3
System.out.println("Size of the array: " + size);
// Output: Size of the array: 3
```

How to Print Array Elements?

You can print each element of the array using a loop:

Example:

```
int[] arr = {5, 3, 9};
for (int i = 0; i < arr.length; i++) {
    System.out.print(arr[i] + " ");
    // Outputs: 5 3 9
}
```

How to Take Input from User in an Array?

You can use the `Scanner` class to take user input and store it in an array.

Example:

```
import java.util.Scanner;

public class ArrayInputExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int size = 5; // Size of the array
```

```
// Declaration and creation of the array
        int[] arr = new int[size];

        // Taking input from the user
         System.out.println("Enter " + size + "
integers:");
        for (int i = 0; i < size; i++) {
            arr[i] = scanner.nextInt();
         // Stores user input in the array
        }



        // Printing the array elements
        System.out.println("Array elements are:");
        for (int i = 0; i < size; i++) {
            System.out.print(arr[i] + " ");
         // Outputs the elements of the array
        }
    }
}
```


Key Points to Remember:

- Arrays are fixed in size after creation.
- Use `.length` to determine the number of elements in the array.
- Use a loop to iterate through the elements for printing or processing.
- Java arrays are zero-indexed, meaning the first element is at index 0.

For-Each Loop

What is a For-Each Loop?

- The **for-each loop** (also known as the **enhanced for loop**) in Java provides a simpler way to iterate through elements of an array or a collection without needing to use an index. It is particularly useful for working with arrays, as it makes the code cleaner and easier to read.

Syntax:

```
for (dataType element : arrayNameOrCollection) {
    // Code to execute for each element
}
```

Example:

```
public class ForEachExample {
    public static void main(String[] args) {
        int[] arr = {5, 3, 9, 1, 4};
        // Using a for-each loop to print array elements
        for (int element : arr) {
            System.out.print(element + " ");
          // Outputs: 5 3 9 1 4
        }
    }
}
```

Advantages of For-Each Loop

1. **Simplicity**:
    - The for-each loop makes code easier to read and write. You don't have to worry about the size of the array or managing an index variable.
2. **Less Error-Prone**:

- Since you don’t manually handle the loop index, there are fewer chances of errors, such as off-by-one errors (e.g., accessing an index that is out of bounds).

3. **Cleaner Code**:
   - The code looks cleaner and focuses more on the logic of what you want to achieve rather than how you are iterating through the collection.

Limitations of For-Each Loop

1. **Modification of Elements**:
   - You cannot change the elements of the array or collection directly within the for-each loop. If you need to modify the elements, you would need a traditional for loop.

```
for (int element : arr) {
    element = element * 2;
// This won't change the actual elements in the array
}
```

2. **No Access to Index**:
   - You do not have direct access to the index of the current element in a for-each loop. If you need to know the index, you must use a regular for loop.

What is an Object Array?

An object array is an array that holds multiple objects. These objects are instances of a class. Instead of storing primitive data types (like `int`, `float`, etc.), the array stores **references** to objects.

Steps to Store Objects in an Array:

1. **Define a Class**: First, you define a class that will serve as the blueprint for the objects you will create and store in the array.
2. **Declare an Array of Objects**: You declare an array that will hold references to objects of the class you created.
3. **Create Objects and Store in the Array**: You create instances of the class using the `new` keyword and assign them to the array.
4. **Access and Use Objects**: Once the objects are stored in the array, you can access and manipulate them using their index.

Example: Creating and Storing `Student` Objects in an Array

Let's walk through the example of creating and storing objects of a class called `Student` in an array.

Step 1: Create the `Student` Class

```
class Student {
    String name;
    int age;

    // Constructor to initialize the Student object
    Student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Method to display student details
```

```
    void display() {
        System.out.println("Name: " + name + ",
Age: " + age);
    }
}
```

Here, the `Student` class has two properties (`name` and `age`) and a constructor to initialize these properties when a `Student` object is created. The `display()` method prints the student details.

Step 2: Declare and Create an Array of Objects

Now, let's declare and initialize an array that can store multiple `Student` objects:

```
public class Main {
    public static void main(String[] args) {
        // Step 1: Declare an array to hold 3 Student
objects
        Student[] students = new Student[3];

        // Step 2: Create Student objects and store them
in the array
        students[0] = new Student("Alice", 20);
// First Student object
        students[1] = new Student("Bob", 22);
// Second Student object
        students[2] = new Student("Charlie", 21);
// Third Student object

        // Step 3: Access and display each Student
object
        for (int i = 0; i < students.length; i++)
{
            students[i].display();
// Calling display method on each object
```

```
        }
    }
}
```

Key Points to Remember:

1. **Array Holds References to Objects**:
    - The array `students[]` holds references to `Student` objects, not the actual objects themselves. Each element in the array points to an object stored in memory.
2. **Creating Objects**:
    - Each object in the array is created using the `new` keyword, like `new Student("Alice", 20);`. This creates an object in memory, and the reference to this object is stored in the array.
3. **Fixed Array Size**:
    - The array size is fixed when it is declared. In the example, the array can hold exactly 3 `Student` objects. You cannot add more objects to this array once it's full.
4. **Accessing Objects**:
    - You can access and use the objects stored in the array by their index, like `students[0]`. From here, you can call the object's methods (like `display()`) or access its fields (like `name` or `age`).

When to Use an Array of Objects?

- **Organized Data Storage**: Arrays of objects are useful when you want to store multiple objects of the same type in a structured way. For example, storing multiple `Student` objects in an array helps manage student data effectively.
- **Fixed Number of Objects**: If you know in advance how many objects you need to store (like a fixed number of students), an array is a good option. However, if you need a dynamic-sized collection, you should use an `ArrayList`.

Multi Dimensional Array

- multi-dimensional array is an array that contains more than one dimension. This means you can have an array inside another array, commonly represented as a matrix or a table.

Two-Dimensional Arrays in Java

Definition:
A two-dimensional array, often referred to as a **matrix**, is an array organized in rows and columns. Each element in a 2D array is identified by its row and column indices.

Characteristics:

- A 2D array in Java is essentially an **array of arrays**, meaning it is a collection of similar data types stored in contiguous memory locations.
- It stores **homogeneous data**, meaning all elements must be of the same data type.

1. How to Declare a 2D Array

Syntax:

```
int[][] arr = new int[2][3];
// A 2D array with 2 rows and 3 columns
```

- **int[][]**: Data type of the array (an array of integers).
- **arr**: Name of the array.
- **new int[2][3]**: Allocates memory for a 2D array with 2 rows and 3 columns.

You can also declare a 2D array with variables for rows and columns:

```
int rows = 4;
int columns = 4;
int[][] arr = new int[rows][columns];//A 4x4 2dArray
```

2. How to Initialize a 2D Array

You can initialize a 2D array in several ways:

Method 1: Element-wise Initialization

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

Method 2: Declare and Initialize Together

```
int[][] arr = {
    {1, 4, 2},   // First row
    {3, 6, 8}    // Second row
};
// Alternatively, if you want to specify an extra row:
int[][] arr = {
    {1, 4, 2},
    {3, 6, 8},
    {9, 10, 5}  // Third row
};
```


3. How to Print 2D Array Elements

You can access and print the elements of a 2D array using nested loops:

```
int[][] arr = {
```

```
    {1, 4},
    {3, 6}
};

// Printing elements
System.out.print(arr[0][0] + " "); // Output: 1
System.out.print(arr[0][1] + " "); // Output: 4
System.out.print(arr[1][0] + " "); // Output: 3
System.out.print(arr[1][1] + " "); // Output: 6
```

4. How to Take Input from User in a 2D Array

You can take user input using nested loops as follows:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        int rows = 3;
        int columns = 4;
        int[][] arr = new int[rows][columns];

        // Taking input from the user
        System.out.println("Enter elements for the
2D array:");
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
                arr[i][j] = scanner.nextInt(); //
Store input in arr[i][j]
            }
        }

        scanner.close(); // Close the scanner
```

```
    }
}
```

5. How to Print 2D Array Elements after User Input

To print the elements after taking input, you can use the following nested loops:

```
// After taking input
System.out.println("The 2D array elements are:");
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < columns; j++) {
        System.out.print(arr[i][j] + " "); // Print each
element
    }
    System.out.println(); // Move to the next line after
each row
}
```

Anonymous array

an **anonymous array** in Java is an array that is declared without explicitly assigning it a reference variable. Instead, it is passed directly to a method or used immediately in the code. This technique is useful when you need to quickly pass an array to a method without the need to retain the array's reference after its usage.

Key Characteristics:

- No reference variable is assigned to the array.
- It is typically used when the array is required temporarily.

- Commonly passed as an argument to methods.

Syntax of Anonymous Array:

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

Example of Anonymous Array Usage:

```
public class AnonymousArrayExample {
    // A method that takes an array as input and
prints the sum of its elements
    static void sumOfElements(int[] numbers) {
        int sum = 0;
        for (int num : numbers) {
            sum += num;
        }
        System.out.println("Sum: " + sum);
    }

    public static void main(String[] args) {
        // Using an anonymous array as an argument
        sumOfElements(new int[]{5, 10, 15, 20});
    }
}
```

Explanation:

- In the above example, the array `new int[]{5, 10, 15, 20}` is an anonymous array.
- It is directly passed to the `sumOfElements` method without being assigned to any variable.
- This array will exist only during the execution of the method, and its reference is not stored anywhere.

Advantages of Anonymous Array:

- **Short-lived usage**: If you only need the array for a method call or short operation, it saves the overhead of declaring a variable.
- **Cleaner code**: For one-time array operations, using anonymous arrays can make the code more concise.

---

jagged array

A **jagged array** in Java is a multi-dimensional array where the rows (sub-arrays) can have different lengths. Unlike a regular 2D array where every row has the same number of columns, a jagged array allows you to have rows of varying sizes, making it more flexible.

Key Characteristics:

- Each row is an array and can have its own size.
- It is useful when you need a structure where rows have different lengths.

How to Use a Jagged Array:

1. Declaration: Declare an array with a fixed number of rows but leave columns undefined.

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

2. Initialization: For each row, define the size of the sub-array (column length).

```
jaggedArray[0] = new int[2]; // Row 1 with 2 columns
jaggedArray[1] = new int[4]; // Row 2 with 4 columns
```

3. Accessing Elements: You can access elements like a regular 2D array, but each row might have a different length.

```
jaggedArray[1][2] = 5; // Assigning a value
```

4. Iteration: Use nested loops to iterate over the array. Be mindful of the different row lengths.

```
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();
     }
```

Example of a Jagged Array in Java:

```
public class JaggedArrayExample {
    public static void main(String[] args) {
        // Declare a 2D jagged array with 3 rows
        int[][] jaggedArray = new int[3][];
        // Assign different lengths to each row
        jaggedArray[0] = new int[2];  // First row
with 2 columns
```

```
        jaggedArray[1] = new int[4];  // Second row
with 4 columns

        jaggedArray[2] = new int[3];  // Third row
with 3 columns

        // Initialize the array with values

        jaggedArray[0][0] = 1;

        jaggedArray[0][1] = 2;

        jaggedArray[1][0] = 3;

        jaggedArray[1][1] = 4;

        jaggedArray[1][2] = 5;

        jaggedArray[1][3] = 6;

        jaggedArray[2][0] = 7;

        jaggedArray[2][1] = 8;

        jaggedArray[2][2] = 9;

        // Print the 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(); // Move to the
next line after each row
```

```
            }

        }

}
```