9.2-Creation of Array

Introduction

An array is a collection of elements of the same type, stored in contiguous memory locations. In Java, an array is an object that can hold multiple values of the same data type. This data structure is useful when you need to store a fixed set of elements, as it allows efficient data storage and retrieval.

Advantages of Arrays

- **Code Optimization:** Arrays help optimize code by allowing efficient data retrieval and sorting.
- **Random Access:** You can directly access any element in an array using its index, making data manipulation straightforward.

Disadvantages of Arrays

- **Size Limit:** Arrays have a fixed size; once declared, their size cannot change during runtime. To overcome this limitation, Java provides a collection framework that allows dynamic resizing of data structures.

Types of Arrays in Java

There are two main types of arrays in Java:

1. **Single-Dimensional Array:** A linear array with elements stored in a single row.
2. **Multidimensional Array:** An array of arrays, where elements are stored in a matrix form, like a table with rows and columns.

Syntax to Declare an Array in Java

You can declare an array in Java using any of the following syntaxes:

```
dataType[] arr;  // Example: int[] arr;

dataType []arr;  // Example: int []arr;

dataType arr[];  // Example: int arr[];
```

Implementation of Arrays

Example 1: Declaring and Initializing an Array

If you know the number of elements you want to store, you can directly initialize the array as follows:

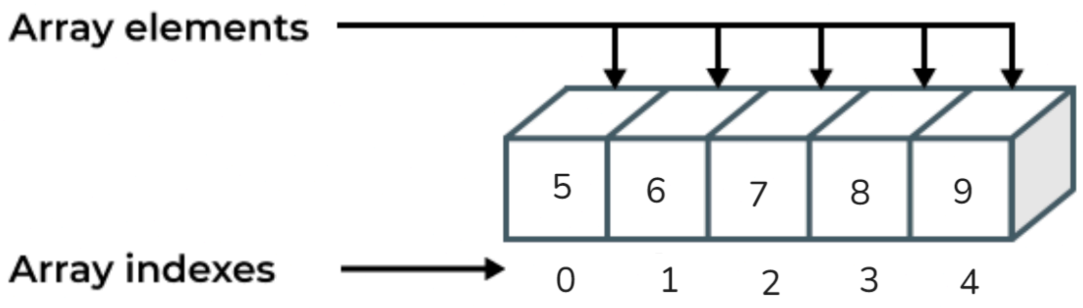
```
int num[] = {5, 6, 7, 8, 9};
```

Alternatively, you can declare the array first and then allocate memory:

`int num[] = new int[4];` // This creates an array with 4 elements, initially set to the default value (0 for int).

Accessing Array Elements

In arrays, elements are stored in indexed positions, starting from 0. For example:



One-dimensional array with five elements

```
System.out.println(num[0]);  // Output: 5
```

This code will print the value at index 0, which is 5. You can access other elements similarly.

Example 2: Updating an Array Element

Suppose you want to change the value at index 1 (currently 6) to 0:

```
num[1] = 0;

System.out.println(num[1]);  // Output: 0
```

This updates the value at index 1 to 0 and then prints it.

Example 3: Working with Arrays of Unknown Size

If you don't know how many values you need to store initially, you can declare an array and allocate memory for a fixed number of elements:

```
int nums[] = new int[4];  // Allocates memory for 4 elements, all initially set to 0.
```

Later, you can assign values to these elements:

```
nums[0] = 3;

nums[1] = 4;

nums[2] = 5;

nums[3] = 6;
```

You can print the values individually:

```
System.out.println(nums[0]);  // Output: 3

System.out.println(nums[1]);  // Output: 4

System.out.println(nums[2]);  // Output: 5

System.out.println(nums[3]);  // Output: 6
```

Example 4: Using a Loop to Access Array Elements

Instead of manually accessing each element, you can use a loop to make the task easier, especially when dealing with larger arrays:

```
for(int i = 0; i <= 3; i++) {

    System.out.print(nums[i] + " ");  // Output: 3 4 5 6

}
```

This loop prints all the elements in the array in a single line.