**Arrays in Java**

In Java, an array is an important linear data structure that allows us to store multiple values of the same type.

* Arrays in Java are objects, like all other objects in Java, arrays implicitly inherit from the java.lang.Object class. This allows you to invoke methods defined in Object (such as toString(), equals() and hashCode()).
* Arrays have a built-in length property, which provides the number of elements in the array

public class Geeks {

public static void main(String[] args) {

​

// initializing array

int[] arr = { 40,55,63,17,22,68,89,97,89};

​

// size of array

int n = arr.length;

​

// traversing array

for (int i = 0; i < n; i++)

System.out.print(arr[i] + " ");

}

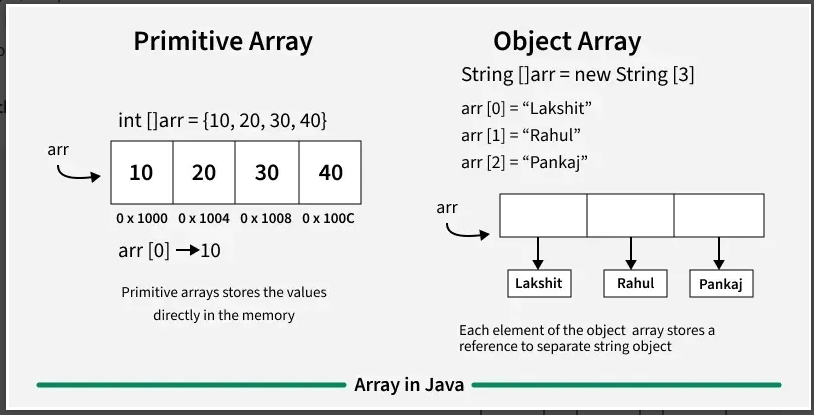
}

**Output**

40 55 63 17 22 68 89 97 89

**Key features of Arrays**

* **Store Primitives and Objects:** Java arrays can hold both primitive types (like int, char, boolean, etc.) and objects (like String, Integer, etc.)
* **Contiguous Memory Allocation**When we use arrays of primitive types, the elements are stored in contiguous locations. For non primitive types, references of items are stored at contiguous locations.
* **Zero-based Indexing:** The first element of the array is at index 0.
* **Fixed Length:**After creating an array, its size is fixed; we can not change it.

****

**Basics Operation on Arrays in Java**

**1. Declaring an Array**

The general form of array declaration is

*// Method 1:  
int arr[];*

*// Method 2:  
int[] arr;*

The element type determines the data type of each element that comprises the array. Like an array of integers, we can also create an array of other primitive data types like char, float, double, etc. or user-defined data types (objects of a class).

***Note:*** *It is just how we can create is an array variable, no actual array exists. It merely tells the compiler that this variable (int Array) will hold an array of the integer type.*

**2. Initialization an Array in Java**

When an array is declared, only a reference of an array is created. We use new to allocate an array of given size.

*int arr[] = new int[size];*

* Array Declaration is generally static, but if the size in not defined, the Array is Dynamically sized.
* Memory for arrays is always dynamically allocated (on heap segment) in Java. This is different from C/C++ where memory can either be statically allocated or dynamically allocated.
* The elements in the array allocated by new will automatically be initialized to zero (for numeric types), false (for boolean) or null (for reference types).

**Array Literal in Java**

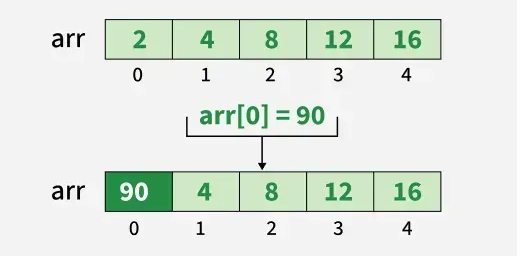
In a situation where the size of the array and variables of the array are already known, array literals can be used.

*// Declaring array literal   
int[] arr = new int[]{ 1,2,3,4,5,6,7,8,9,10 };*

* The length of this array determines the length of the created array.
* There is no need to write the new int[] part in the latest versions of Java.

**3. Change an Array Element**

To change an element, assign a new value to a specific index. The index begins with 0 and ends at (total array size)-1.



*// Changing the first element to 20  
arr[0] = 90;*

**4. Array Length**

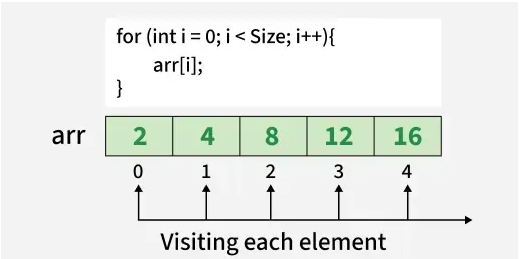
We can get the length of an array using the length property:

*// Getting the length of the array  
int n = arr.length;*

**5. Accessing and Updating All Array Elements**

* All the elements of array can be accessed using Java for Loop.
* Each element in the array is accessed via its index.

Java program to illustrate creating an array of integers, puts some values in the array and prints each value to standard output



public static void main(String[] args)

{

// declares an Array of integers.

int[] arr;

​

// allocating memory for 5 integers.

arr = new int[5];

​

// initialize the elements of the array, first to last(fifth) element

arr[0] = 2;

arr[1] = 4;

arr[2] = 8;

arr[3] = 12;

arr[4] = 16;

​

// accessing the elements of the specified array

for (int i = 0; i < arr.length; i++)

System.out.println("Element at index " + i + " : " + arr[i]);

}

}

**Output**

Element at index 0 : 2

Element at index 1 : 4

Element at index 2 : 8

Element at index 3 : 12

Element at index 4 : 16

**Arrays of Objects in Java**

An array of objects is created like an array of primitive-type data items

**Example**: Here we are taking a student class and creating an array of Student with five Student objects stored in the array. The Student objects have to be instantiated using the constructor of the Student class and their references should be assigned to the array elements.

class Student {

public int roll\_no;

public String name;

Student(int roll\_no, String name){

this.roll\_no = roll\_no;

this.name = name;

}

}

​

public class Geeks {

public static void main(String[] args){

// declares an Array of Student

Student[] arr;

​

// allocating memory for 5 objects of type Student.

arr = new Student[5];

​

// initialize the elements of the array

arr[0] = new Student(1, "aman");

arr[1] = new Student(2, "vaibhav");

arr[2] = new Student(3, "shikar");

arr[3] = new Student(4, "dharmesh");

arr[4] = new Student(5, "mohit");

​

// accessing the elements of the specified array

for (int i = 0; i < arr.length; i++)

System.out.println("Element at " + i + " : { "

+ arr[i].roll\_no + " "

+ arr[i].name+" }");

}

}

**Output**

Element at 0 : { 1 aman }

Element at 1 : { 2 vaibhav }

Element at 2 : { 3 shikar }

Element at 3 : { 4 dharmesh }

Element at 4 : { 5 mohit }

**What happens if we try to access elements outside the array size?**

JVM throws ArrayIndexOutOfBoundsException to indicate that the array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of an array.

Below code shows what happens if we try to access elements outside the array size:

public class Geeks {

public static void main(String[] args)

{

int[] arr = new int[4];

arr[0] = 10;

arr[1] = 20;

arr[2] = 30;

arr[3] = 40;

​

System.out.println(

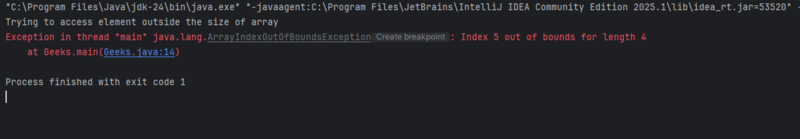
"Trying to access element outside the size of array");

System.out.println(arr[5]);

}

}

Output:

Output of elements outside the array size

**Passing Arrays to Methods**

Like variables, we can also pass arrays to methods. For example, the below program passes the array to method sum to calculate the sum of the array's values.

**Example:**

public class Geeks {

// Driver method

public static void main(String args[])

{

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

​

// passing array to method m1

sum(arr);

}

​

public static void sum(int[] arr)

{

// getting sum of array values

int sum = 0;

​

for (int i = 0; i < arr.length; i++)

sum += arr[i];

​

System.out.println("sum of array values : " + sum);

}

}

**Output**

sum of array values : 15

**Explanation**

* This Java program demonstrates how to pass an array to a method.
* An integer array arr is declared and initialized in the main method.
* The sum() method is called with arr as an argument.
* Inside the sum() method, all array elements are added using a for loop.
* The final sum is then printed to the console.

**Returning Arrays from Methods**

As usual, a method can also return an array. For example, the below program returns an array from method m1.

**Example:**

class Geeks {

// Driver method

public static void main(String args[])

{

int arr[] = m1();

​

for (int i = 0; i < arr.length; i++)

System.out.print(arr[i] + " ");

}

​

public static int[] m1()

{

// returning array

return new int[] { 1, 2, 3 };

}

}

**Output**

1 2 3

**Advantages of Java Arrays**

* **Efficient Access:** Accessing an element by its index is fast and has constant time complexity, O(1).
* **Memory Management:**Arrays have fixed size, which makes memory management straightforward and predictable.
* **Data Organization:** Arrays help organize data in a structured manner, making it easier to manage related elements.

**Disadvantages of Java Arrays**

* **Fixed Size:** Once an array is created, its size cannot be changed, which can lead to memory waste if the size is overestimated or insufficient storage if underestimated.
* **Type Homogeneity:** Arrays can only store elements of the same data type, which may require additional handling for mixed types of data.
* **Insertion and Deletion:** Inserting or deleting elements, especially in the middle of an array, can be costly as it may require shifting elements.