Arrays, Arraylist & Vectors

Day 34
13/08/2025

Why these Data structures?

The need for arrays, Arraylist, Vector and similar data structures comes from a very practical problem in programming - we rarely work with just a single value.

We often need to store, manage & process collections of data in an efficient, organized way.

Q) Why can't we use individual variables?

Imagine you are writting a aprogram to store marks of 100 students.

Memory

Mark 1 = 19
Mark 2 = 21
Mark 3 = 18
Mark 4 = 13
⋮
Mark 100 = 08

Problems

We Have to Create 100 Variables
Managing the 100 Variables is Hard
Issue with scalabilty
What if student Number is 200?

Why Arrays?

An Array is a fixed-size, contiguous block of Memory that stores elements of the same type.

Advantages

- Fast Random access (O(1)) using indexes.
- Memory efficient for known-size collections

Limitations

- Fixed size - Can't grow / shrink dynamically.
- Insertion / removal in the Middle is

costly (O(n))

Why ArrayList?

An Arraylist is resizable array In Java

- Advantages
  1. Dynamically grows / shrinks
  2. provides useful built-in methods (add, remove, contains)

- Limitations
  1. Not synchronized (not thread-safe) by default
  2. slightly more overhead then Arrays.

Why Vectors?

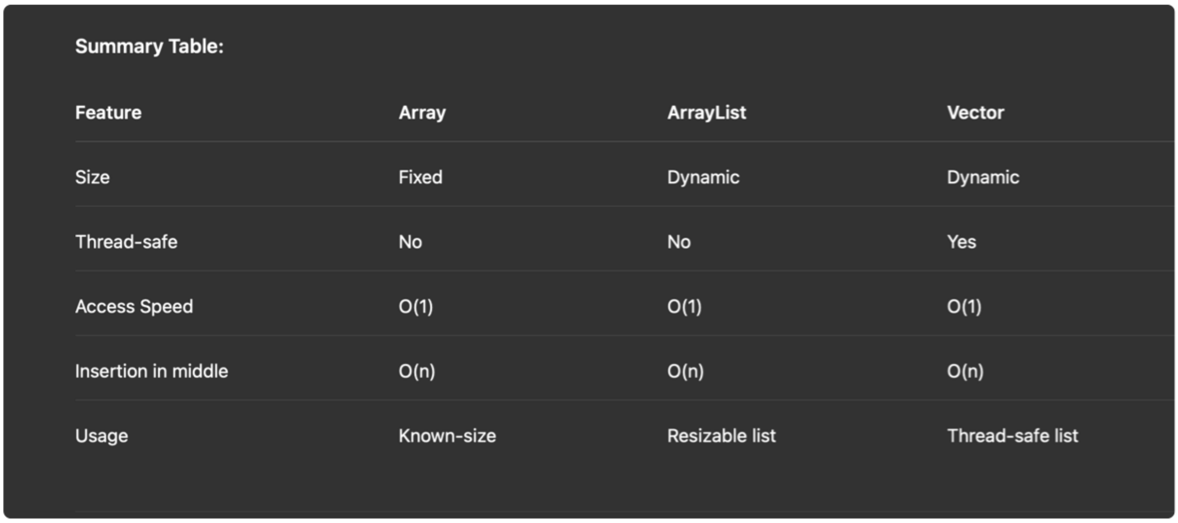
A vector is similar to Arraylist But synchronized

- Advantages

  Thread Safe - can be shared between threads without Extra synchronization

- Limitations

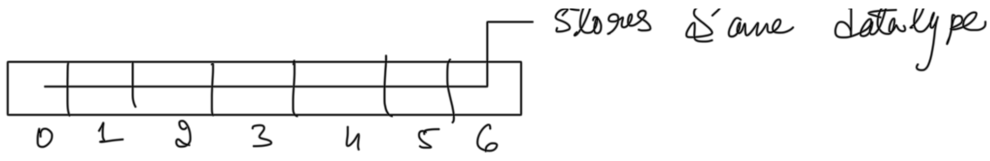
  Slower then array list due to synchronization overhead.



Summary Table:

Feature	Array	ArrayList	Vector
Size	Fixed	Dynamic	Dynamic
Thread-safe	No	No	Yes
Access Speed	O(1)	O(1)	O(1)
Insertion in middle	O(n)	O(n)	O(n)
Usage	Known-size	Resizable list	Thread-safe list

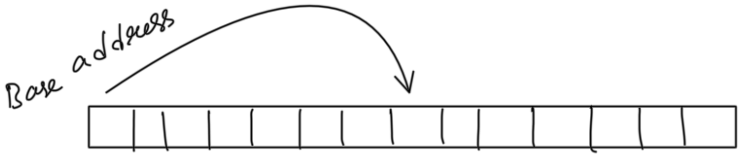
Array

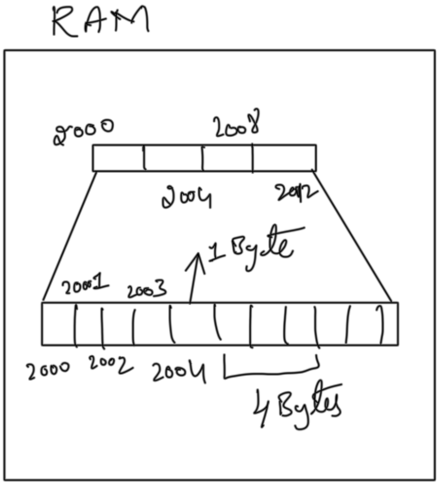


Array Have the Indexing concept which is used to Access the particular indexed Element.

The Index Always starts from 0

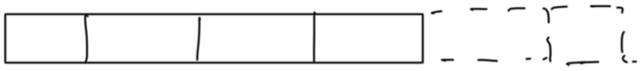
Consider a Array of Integer type





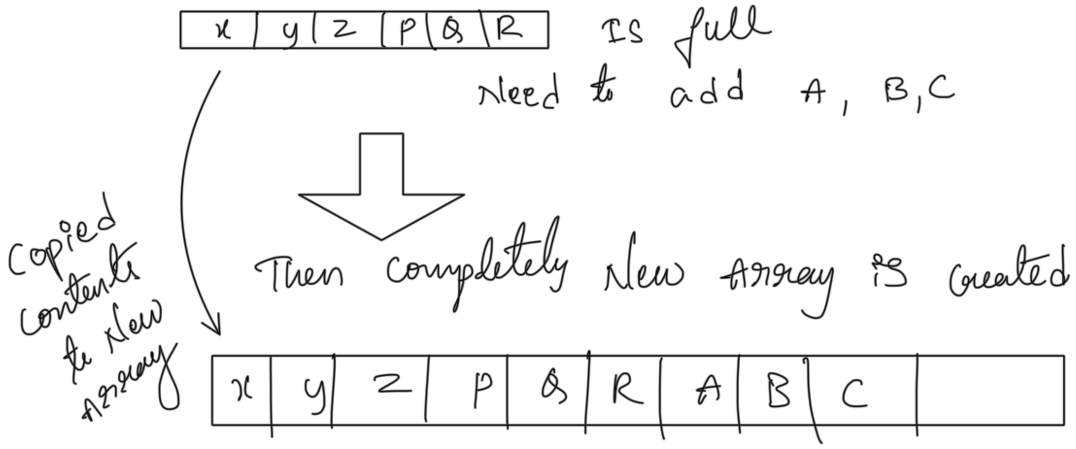
With the Help of Base address Any Element can be accessed.

But array data structure is of fixed size



It cannot grow or shrink.

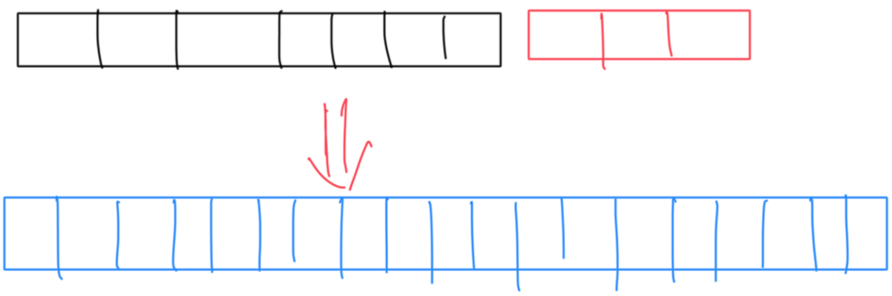
C++ → Vectors
Python → list
JAVA → Array list
} Dynamically grow



Is full
Need to add A, B, C

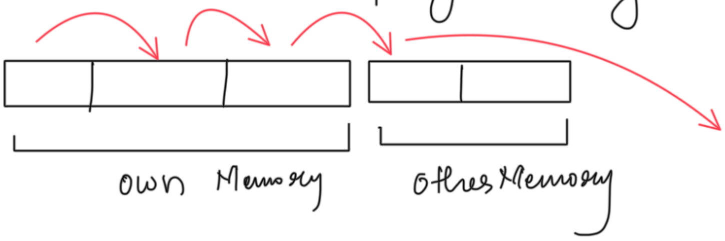
Then completely New Array is created

When the Data structure Become full and Needs to add the Element to it then it asks RAM for 2 times the New Memory and copies old array elements and New elements which Need to be added.



Takes Extra Memory to add more elements in future.

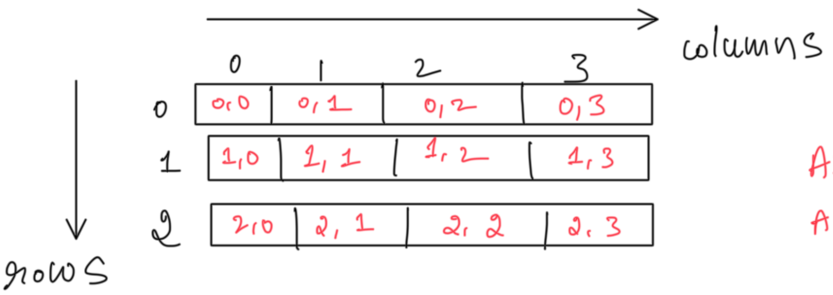
When it comes to C programming.



It doest throws any Index out of Bound Errors The pointer Increments and goes to unknown memory

But C++, JAVA, python throws Exception of Index out of Bound.

Multi Dimentional Arrays



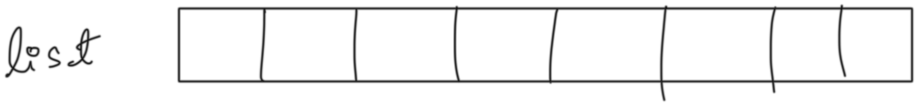
	0	1	2	3
0	0,0	0,1	0,2	0,3
1	1,0	1,1	1,2	1,3
2	2,0	2,1	2,2	2,3

Array [0] [0]

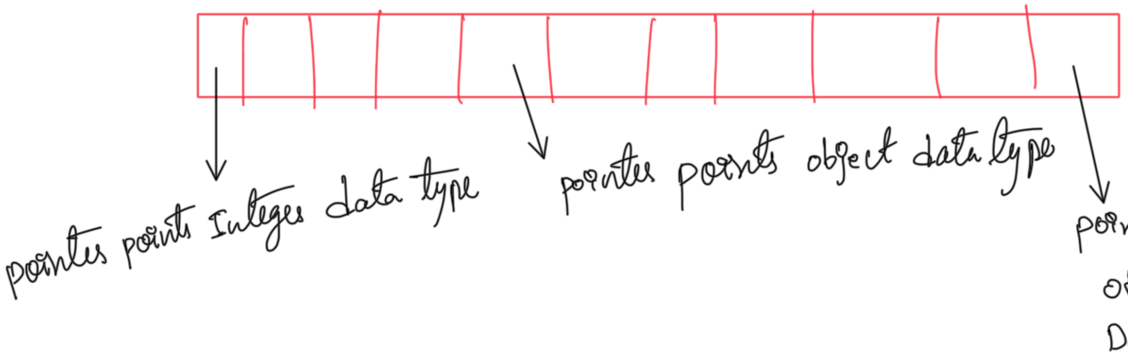
Array [x] [y]

row column

list in python is Heterogenous data type.



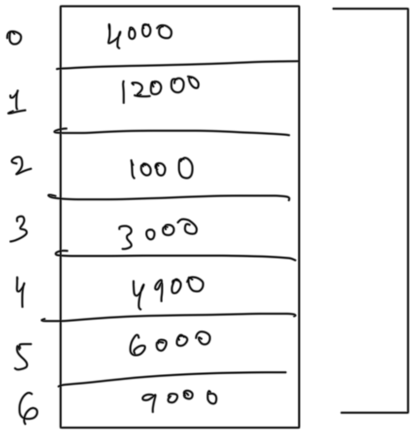
In the list Instead of storing data it stores the pointer ! the pointer Points to different Memory address of different data types


pointer points Integer data type

pointer points object data type

pointer points other type of Data.

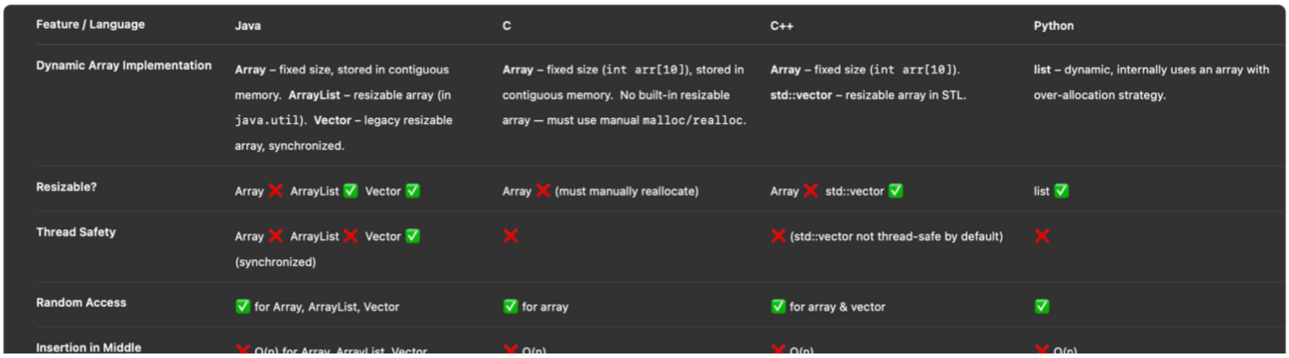
With the Help of pointers instead of Remembering the Values We Just remembers the pointer (address) The data Structure which stores pointers



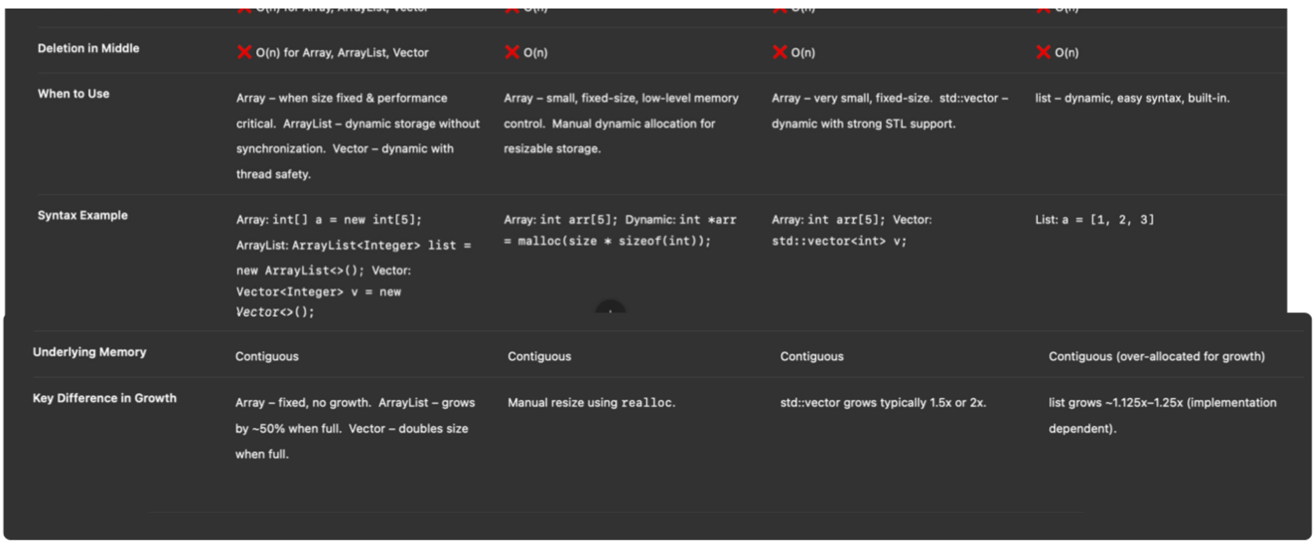
0	4000
1	12000
2	1000
3	3000
4	4900
5	6000
6	9000

Array of type Address

The Array Contain address which points to different data types stored at some address of Memory.

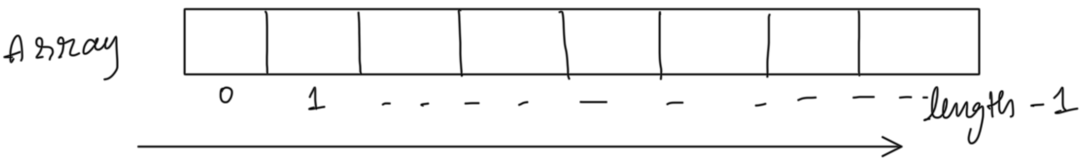


Feature / Language	Java	C	C++	Python
Dynamic Array Implementation	**Array** – fixed size, stored in contiguous memory. **ArrayList** – resizable array (in `java.util`). **Vector** – legacy resizable array, synchronized.	**Array** – fixed size (`int arr[10]`), stored in contiguous memory. No built-in resizable array — must use manual `malloc/realloc`.	**Array** – fixed size (`int arr[10]`). **std::vector** – resizable array in STL.	**list** – dynamic, internally uses an array with over-allocation strategy.
Resizable?	Array ❌ ArrayList ✅ Vector ✅	Array ❌ (must manually reallocate)	Array ❌ std::vector ✅	list ✅
Thread Safety	Array ❌ ArrayList ❌ Vector ✅ (synchronized)	❌	❌ (std::vector not thread-safe by default)	❌
Random Access	✅ for Array, ArrayList, Vector	✅ for array	✅ for array & vector	✅
Insertion in Middle	❌ O(n) for Array, ArrayList, Vector	❌ O(n)	❌ O(n)	❌ O(n)



	❌ O(n) for Array, ArrayList, Vector	❌ O(n)	❌ O(n)	❌ O(n)
Deletion in Middle	❌ O(n) for Array, ArrayList, Vector	❌ O(n)	❌ O(n)	❌ O(n)
When to Use	Array – when size fixed & performance critical. ArrayList – dynamic storage without synchronization. Vector – dynamic with thread safety.	Array – small, fixed-size, low-level memory control. Manual dynamic allocation for resizable storage.	Array – very small, fixed-size. std::vector – dynamic with strong STL support.	list – dynamic, easy syntax, built-in.
Syntax Example	Array: `int[] a = new int[5];` ArrayList: `ArrayList<Integer> list = new ArrayList<>();` Vector: `Vector<Integer> v = new Vector<>();`	Array: `int arr[5];` Dynamic: `int *arr = malloc(size * sizeof(int));`	Array: `int arr[5];` Vector: `std::vector<int> v;`	List: `a = [1, 2, 3]`
Underlying Memory	Contiguous	Contiguous	Contiguous	Contiguous (over-allocated for growth)
Key Difference in Growth	Array – fixed, no growth. ArrayList – grows by ~50% when full. Vector – doubles size when full.	Manual resize using `realloc`.	std::vector grows typically 1.5x or 2x.	list grows ~1.125x–1.25x (implementation dependent).

Accessing Array elements / Traversing on Array.



-1 Beause Index starts from 0.

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

        Array [index]
      // forward traversing
}
```

Backword traversing

```
for (int index = arraylength-1 ; i >=0 ; i--) {

        Array[index]

}
```

Array [Expression]

Expression can be written in the Square Brackets

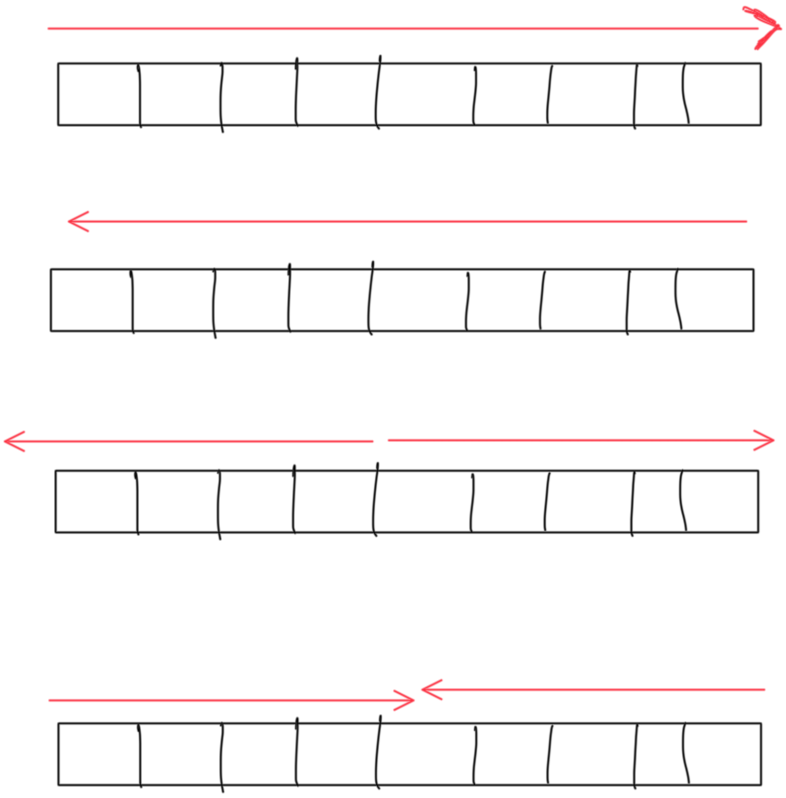
Ext Index+1 index-2

Index++ -- Index

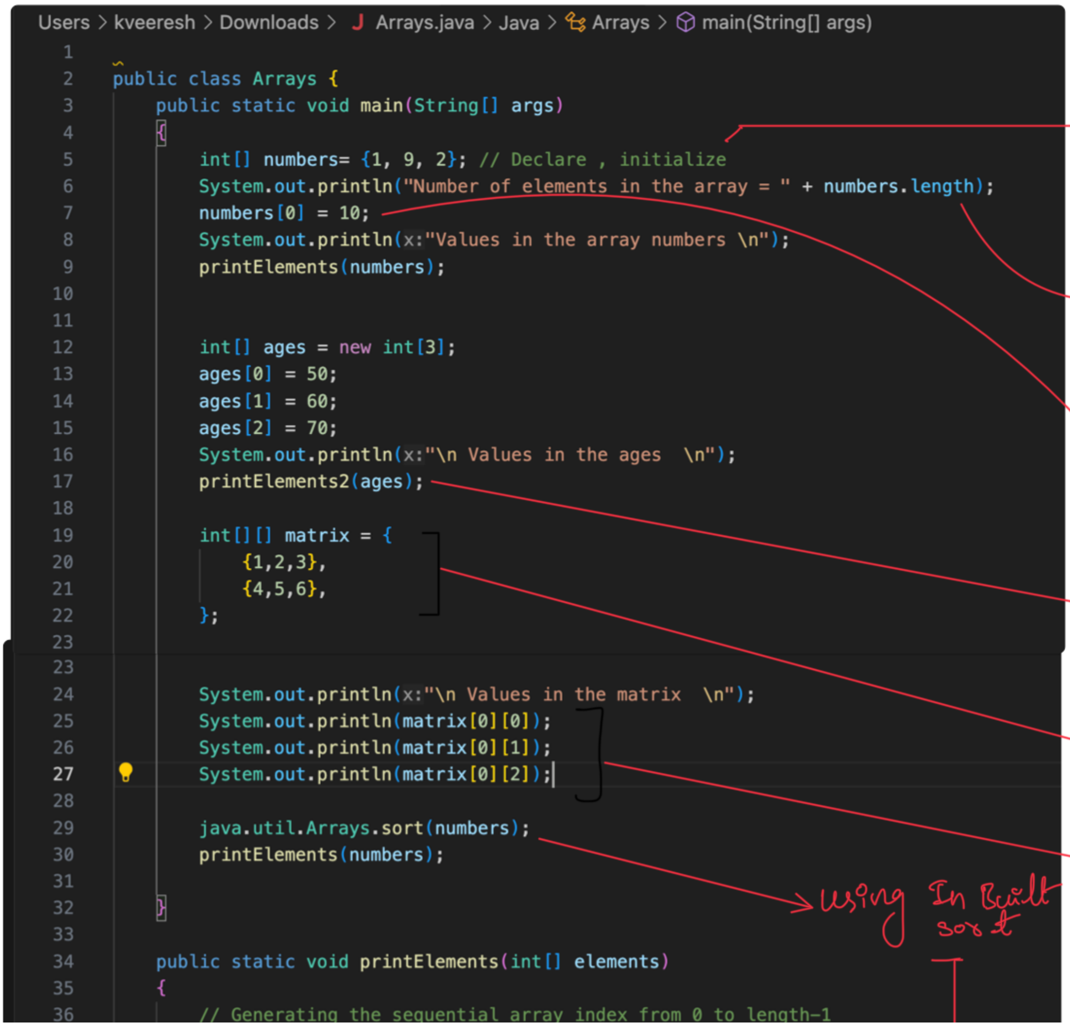
Index-- ++ Index

Index*2 Index/2

It's An Important feature of Array.



Traversing is possible In Any direction on the array Elements.



Users > kveeresh > Downloads > J Arrays.java > Java > Arrays > main(String[] args)

```
public class Arrays {
    public static void main(String[] args)
    {
        int[] numbers= {1, 9, 2}; // Declare , initialize
        System.out.println("Number of elements in the array = " + numbers.length);
        numbers[0] = 10;
        System.out.println(x:"Values in the array numbers \n");
        printElements(numbers);


        int[] ages = new int[3];
        ages[0] = 50;
        ages[1] = 60;
        ages[2] = 70;
        System.out.println(x:"\n Values in the ages  \n");
        printElements2(ages);

        int[][] matrix = {
            {1,2,3},
            {4,5,6},
        };


        System.out.println(x:"\n Values in the matrix  \n");
        System.out.println(matrix[0][0]);
        System.out.println(matrix[0][1]);
        System.out.println(matrix[0][2]);

        java.util.Arrays.sort(numbers);
        printElements(numbers);

    }

    public static void printElements(int[] elements)
    {
        // Generating the sequential array index from 0 to length-1
```

declaring and initilizing

printing length of array

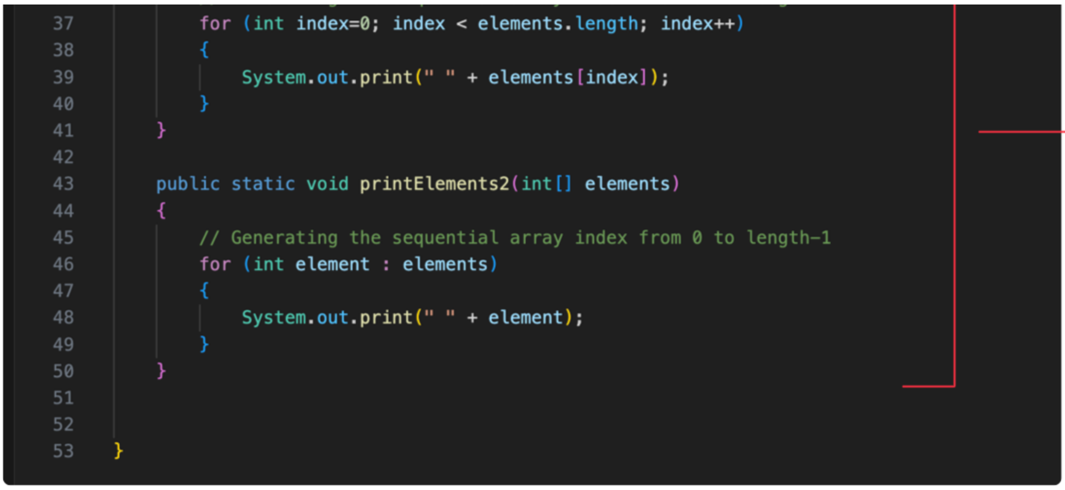
assigning Some Value at Index

printing array

2D array

printing 2D array Elements

using In Built sort



```java
        for (int index=0; index < elements.length; index++)
        {
            System.out.print(" " + elements[index]);
        }
    }

    public static void printElements2(int[] elements)
    {
        // Generating the sequential array index from 0 to length-1
        for (int element : elements)
        {
            System.out.print(" " + element);
        }
    }


}
```

loops for printing Array elements