

# WIX1002

## Fundamentals of Programming

---

### Chapter 5

### Arrays





# Contents

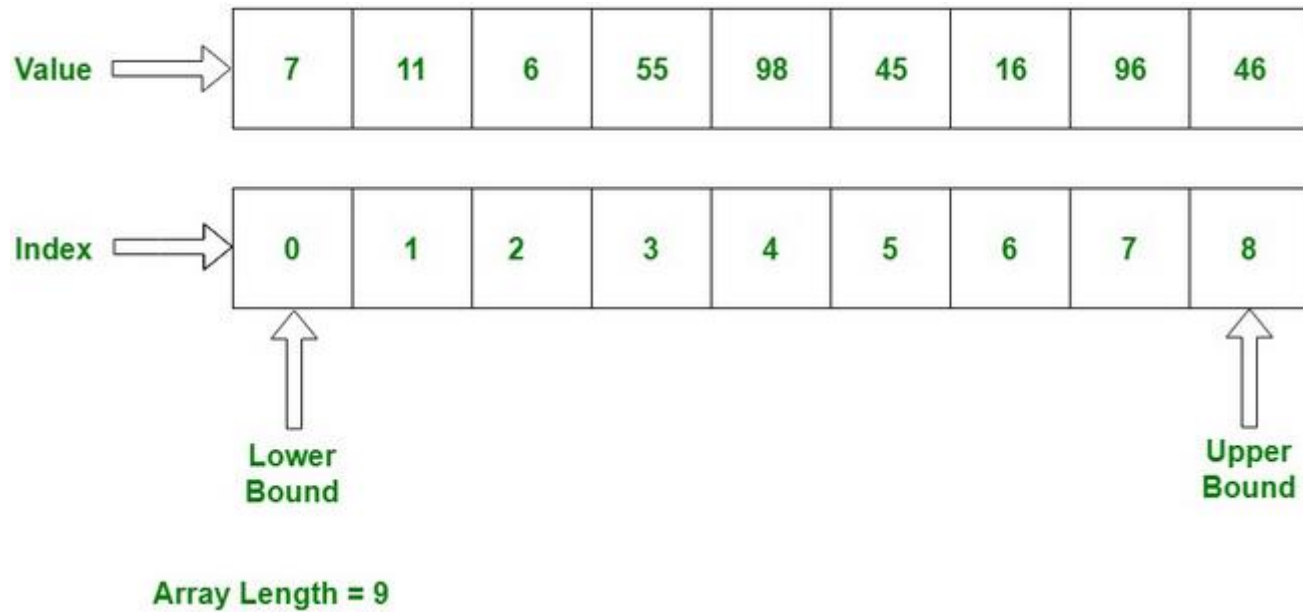
- Introduction
- Multidimensional Arrays
- Applications
- Bubble Sort
- Linear Search
- Binary Search



# Introduction

- An array is a **sequence** of values of the **same type**.
- It is a data structure that used to process a **collection of data** that is from the same type.
- An array is a group of **contiguous memory locations** that all have the **same name and the same type**. It is an ordered list of values and each value has a **numeric index** (subscript)
- An array of size N is indexed from **zero to N-1**
- In Java, an array is a **reference data type**.
- The **new** operator is used to construct the array.

# Introduction





# Introduction

- **Array Declaration**

`typeofArray[] nameOfArray = new typeofArray[length];`

`char[] line = new char[100];`

`double[] score = new double[20];`

- When declare an array, each element of the array receives **a default value zero for the numeric data type, false for boolean and null for references or String.**

- **Array Initialisation**

- `int[] number = {1,2,3,4,5,6};`

- `String[] name = {"one", "two", "three"};`



# Introduction

- A particular value in an array is referenced using the array name followed by the index (subscript) in brackets.
- The first element in every array start with 0.
  - `arrayName[0]` refer to the first element
  - `arrayName[3]` refers to the 4<sup>th</sup> value in the array.
- Every array in Java knows its own length.
  - **`arrayName.length`** - determine the length of the array
  - Java will throw an exception if an array index is **out of bounds**.
- Array elements are not limited to primitive data type but also array of object.



# Introduction

- To display the element of the array
  - `for(int i=0; i<arrayName.length; i++)`
  - `System.out.println(arrayName[i]);`
- Or
  - `for(int value : arrayName)`
  - `System.out.println(value);`
  - However, this method can't be used to modify the value of the element of the array.



# Multidimensional Arrays

- We can declare arrays with more than one index in Java.

- **Multidimensional Arrays Declaration**

```
typeofArray[] ... [] nameOfArray = new  
    typeofArray[length_i] ... [length_n];
```

```
double [][] scoreTable = new double[4][5];
```

//4 rows 5 columns

- Multidimensional Arrays with two subscripts are often used to represent tables of values consisting of information arranged in rows and columns.

- **Multidimensional Arrays Initialisation**

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





# Multidimensional Arrays

- **Ragged Arrays**

- Different rows of the array consists of different columns

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

```
t[0] = new int[3];
```

```
t[1] = new int[10];
```

```
t[2] = new int[5];
```



# Applications

- Summing elements of an array  

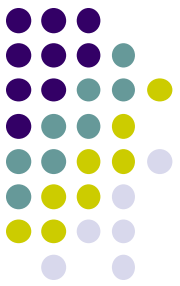
```
int a[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };  
for ( int i = 0; i < a.length; i++ )  
    total += a[ i ];
```
- Sorting is a technique to arrange elements in same order. Sorting data is one of the common feature that can apply on arrays. The common sorting technique is **Bubble Sort**.
- Searching is a technique to locate an item given its key. Searching locating a particular element value in the array. The common searching techniques are **Linear Search** and **Binary Search**.



# BubbleSort

- The bubble sort uses **nested loops** to make several passes through the array. Each pass compares successive pairs of elements.
- If a pair is in increasing order, the bubble sort leaves the value as they are.
- If a pair is in decreasing order, the bubble sort swaps their values in the array.
- It requires several pass over the data.

# BubbleSort



(a) Pass 1

Initial array:

29	10	14	37	13
10	29	14	37	13
10	14	29	37	13
10	14	29	37	13
10	14	29	13	<b>37</b>

(b) Pass 2

10	14	29	13	<b>37</b>
10	14	29	13	<b>37</b>
10	14	29	13	<b>37</b>
10	14	13	<b>29</b>	<b>37</b>



# BubbleSort

```
// control number of passes
for ( int pass = 1; pass < b.length; pass++ )
    // control number of comparison
    for ( int i = 0; i < b.length - 1; i++ )
        if ( b[ i ] > b[ i + 1 ] ) {
            int hold = b[i];
            b[i] = b[i+1];
            b[i+1] = hold;
        }
```



# Linear Search

- This method works well for **small arrays** or for **unsorted arrays**.
- The search technique search the array from the **first position to the last position** in a linear progression.
- It compares each element of the array with a search key.  
for ( int cnt = 0; cnt < arrayName.length; cnt++ )  
    if ( arrayName[ cnt ] == searchKey )  
        return cnt;  
return -1; // key not found



# Binary Search

- This method works well for **large and sorted arrays**.
- It eliminates half of the elements in the array being searched after each comparison.
- It locates the middle array element and compares it to the search key. If the search key not found, the binary search reduces the problem to **search for half** of the array.

```
int low = 0; // low element subscript  
int high = arrayName.length - 1; // high element subscript  
int middle; // middle element subscript
```



# Binary Search

// loop until low subscript is greater than high subscript

```
while ( low <= high ) {  
    middle = ( low + high ) / 2;  
    if ( key == arrayName[ middle ] )  
        return middle;  
    else if ( key < arrayName[ middle ] )  
        high = middle - 1;  
    else  
        low = middle + 1;  
}  
return -1; // key not found
```



