CLASS : Arrays

- **Indexed** collection of fixed number of *homogeneous* elements.
- **Advantage:**
    - Can store `n` elements under a single variable.
- **Disadvantages:**
    - *Fixed* in size.
    - Can contain only *homogeneous* elements.
- **Declaration:** `int x[];`
- **Creation:** `x = new int[3];`
- **Initialization:** `x[0] = 100; x[1] = 50; x[2] = 25;`

Declaration Of Arrays

- **Valid** forms of *Declaration*:
    - `int x[]`, `int []x`
    - `int [][]x`, `int x[][]`, `int []x[]`, `int [] x[]`
- **Invalid** form:
    - `int x[8]` — **size** cannot be *specified* at **declaration**.

Creation of an Array

- Every array in Java is an **object**, hence created using the `new` keyword.

```
int x[] = new int[5];
System.out.println(x.getClass().getName()); // [I
```

- **Size** of an array must be *specified* during **object creation**.

```
int x[];               // valid
x = new int[5];        // valid
int x[] = new int[];   // invalid
```

- Array size cannot be `< 0`.
    - No *compile-time* error.

    - But, `NegativeArraySizeException` is thrown at *run-time*.
- Acceptable data types for **size**: `byte`, `short`, `int`, and `char`.
- Upon creation, arrays are initialized with **default values**.
- Accessing elements outside the valid index range (`0` to `length - 1`) results in `ArrayIndexOutOfBoundsException`.

Array Type → Class Representation

Array Type	Corresponding Class
`int[]`	`[I`
`int[][]`	`[[I`
`byte[]`	`[B`
`short[]`	`[S`
`long[]`	`[J`
`float[]`	`[F`
`double[]`	`[D`
`boolean[]`	`[Z`
`char[]`	`[C`

```java
import java.util.Arrays;
class ArrayExample {
    public static void main(String[] args) {
        int x[] = new int[(short)5];
        long y[] = new long[(byte)6];
        double z[][] = new double['a']['b'];

        x[0] = -9;
        y[1] = -11;
        z[0][2] = -99.89;

        System.out.println(x + " " + y + " " + z);
        System.out.println(Arrays.toString(x));
        System.out.println(Arrays.toString(y));
```

```
15          System.out.println(Arrays.toString(z));
16          System.out.println(Arrays.toString(z[0]));
17          System.out.println(x[5]); // Exception
18      }
19  }
```

2D Arrays

- `int x[][] = new int[1][2];` ≡ `int x[][] = new int[1][]; x[0] = new int[2];`
    - This results in: `x[0][0] = 0; x[0][1] = 0;`
- Also valid:

```
1  int x[][] = new int[3][];
2  x[0] = new int[3];
3  x[1] = new int[5];
4  x[2] = new int[2];
```

- This demonstrates **Multi-Dimensional Arrays**, where each element is a reference to another *array*, ending in data type *literals*.

```
1  import java.util.Arrays;
2  class MultiDimensionalArrays {
3      public static void main(String[] args) {
4  /*  Alternate for: (17-27)
5          int x[][][] = new int[4][][];
6          x[0] = new int [3][]; x[1] = new int [3][]; x[2] = new int
   [3][]; x[3] = new int [3][];
7          x[0][0] = new int[2]; x[0][1] = new int[2]; x[0][2] = new
   int[2];
8          x[1][0] = new int[2]; x[1][1] = new int[2]; x[1][2] = new
   int[2];
9          x[2][0] = new int[2]; x[2][1] = new int[2]; x[2][2] = new
   int[2];
10         x[3][0] = new int[2]; x[3][1] = new int[2]; x[3][2] = new
   int[2];
11         x[0][0][0] = x[0][0][1] = x[0][1][0] = x[0][1][1] = x[0][2]
   [0] = x[0][2][1] =
12         x[1][0][0] = x[1][0][1] = x[1][1][0] = x[1][1][1] = x[1][2]
   [0] = x[1][2][1] =
13         x[2][0][0] = x[2][0][1] = x[2][1][0] = x[2][1][1] = x[2][2]
   [0] = x[2][2][1] =
```

```
        x[3][0][0] = x[3][0][1] = x[3][1][0] = x[3][1][1] = x[3][2][0] = x[3][2][1] = 9;

*/
        int x[][][] = new int[4][][];
        for (int i = 0; i < 4; i++)
            x[i] = new int[3][];
        for (int i = 0; i < 4; i++)
            for (int j = 0; j < 3; j++)
                x[i][j] = new int[2];

        for (int i = 0; i < 4; i++)
            for (int j = 0; j < 3; j++)
                for (int k = 0; k < 2; k++)
                    x[i][j][k] = 9;

        for (int i = 0; i < x.length; i++) {
            for (int j = 0; j < x[i].length; j++)
                System.out.print(Arrays.toString(x[i][j]) + " ");
            System.out.println();
        }
    }
}
```

Searching in Arrays

- **Linear Search**: Search in a *linear* fashion *across* the array, until element is **found** or **End of Array**.
    - Time Complexity: **O(n)**.
- **Binary Search**: This makes the use of a **sorted** array, to decrease the time complexity of searching.
    - Time Complexity: **O(log n)**.

```
import java.util.Scanner;
class Search {
    int arr[];
    Search(int len) { arr = new int[len]; }

    public void bubbleSort() {
        for (int i = 0; i < this.arr.length - 1; i++) {
            boolean swapped = false;
            for (int j = 0; j < this.arr.length - i - 1; j++) {
```

```
                if (this.arr[j] > this.arr[j + 1]) {
                    int temp = this.arr[j];
                    this.arr[j] = this.arr[j + 1];
                    this.arr[j + 1] = temp;
                    swapped = true;
                }
            }
            if (swapped == false)
                break;
        }
    }

    public int binarySearch(int ele) {
        S.bubbleSort();

        int lb = 0, ub = this.arr.length - 1, mid;

        while(lb <= ub) {
            mid = (lb + ub) / 2;
            if(this.arr[mid] == ele)
                return mid;
            else if(this.arr[mid] > ele)
                ub = mid - 1;
            else lb = mid + 1;
        }

        return -1;
    }

    public int linearSearch(int ele) {
        for(int i = 0; i < arr.length; i++) {
            if(ele == arr[i])
                return i;
        }

        return -1;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Search S;
        boolean lS;
        System.out.print("Enter length: ");
        S = new Search(sc.nextInt());

        System.out.println("Enter " + len + " elements:");
```

```
        for(int i = 0; i < S.arr.length; i++)
            S.arr[i] = sc.nextInt();

        System.out.println("Choose:\n\t1. Linear.\n\t2. Binary");
        lS = (sc.nextByte() == 1) ? true : false;

        System.out.print("Enter element to search for: ");
        int index = (lS) ? S.linearSearch(sc.nextInt()) :
S.binarySearch(sc.nextInt());
        if(index >= 0)
            System.out.println("Element found at: " + index);
        else
            System.out.println("Element not found!");
    }
}
```

Sorting in Arrays

```
import java.util.Scanner;
import java.util.Arrays;

class Sorting {
    static int arr[];

    public static void bubbleSort() {
        for (int i = 0; i < arr.length - 1; i++) {
            boolean swapped = false;
            for (int j = 0; j < arr.length - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                    swapped = true;
                }
            }
            if (swapped == false)
                break;
        }
    }

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

        System.out.print("Enter length: ");
```

```
27          arr = new int[sc.nextInt()];
28
29          System.out.println("Enter " + arr.length + " elements:");
30          for(int i = 0; i < arr.length; i++)
31              arr[i] = sc.nextInt();
32
33          System.out.println("1. Bubble");
34          System.out.print("\tEnter Choice: ");
35          switch(sc.nextByte()) {
36              case 1: bubbleSort(); break;
37              case 2: Sort(); break;
38              default: System.out.println("Invalid Choice!!");
39          }
40
41          System.out.println(Arrays.toString(arr));
42      }
43  }
```