Arrays:

Array:

--> Array is an Object, it able to store more than one element of the same Data Type as per indexing, where index values will start from 0 to size-1 .

--> In Java applications, we are able to utilize arrays in the following two ways.

1. Declare and Initialize
2. Declare then Initialize

1. Declare and Initialize:

--> In this approach, we will declare the array and we will provide elements to the array[Initialization] in single instruction.

EX: int[] a = {10,20,30,40,50};

2. Declare then Initialize

--> In this approach, we will declare the array in one instruction and we will initialize the array in the next instructions.

EX: int[] a = new int[5];

a[0] = 10;

a[1] = 20;

a[2] = 30;

a[3] = 40;

a[4] = 50;

--> In Java applications, there are two types of arrays.

1. Single Dimensional Arrays
2. Multi Dimensional Arrays

1. Single Dimensional Arrays

--> It is an array, it able to store all elements in single level or in single dimension or in single row.

--> There are two ways to utilize Single Dimensional arrays in Java applications.

1. Declare and Initialize
2. Declare then Initialize

1. Declare and Initialize:

Syntax: DataType[] refVar = {Val1, Val2, Val3,....Val_n};

EX: int[] a = {10,20,30,40,50};

2. Declare then Initialize:

Syntax: DataType[] refVar = new DataType[Size];

```
refVar[0] = val1;
refVar[1] = val2;
refVar[2] = val3;
-----
-----
refVar[size-1] = val_n;
```

EX: int[] a = new int[5];

```
a[0] = 10;
a[1] = 20;
a[2] = 30;
a[3] = 40;
a[4] = 50;
```

--> In Java applications, when we pass array reference variable as parameter to System.out.println() method then JVM will access internally toString() method , where All arrays are having their own toString() method internally, they are not depending on the Object class toString() method, they will return a string contains the following format.

1. Single Dimensiona Array: [ArrayDataType@RefValue
2. Two Dimensional Array : [[ArrayDataType@RefValue
3. Three Dimensional Array: [[[ArrayDataType@RefValue

EX:

```
package com.durgasoft.app02.test;


public class Test {


	public static void main(String[] args) {
		int[] a = {10,20,30,40,50};
		System.out.println(a);


		float[] f = {11.11f, 22.22f, 33.33f, 44.44f, 55.55f};
		System.out.println(f);


	}


}
```

--> In Java applications, to get length / size / no of elements of an array we have to use a predefined variable in the from of 'length'.

EX:

```
package com.durgasoft.app02.test;

public class Test {

	public static void main(String[] args) {
		int[] a = {10,20,30,40,50};
		System.out.println(a.length);

		float[] f = {11.11f, 22.22f, 33.33f, 44.44f, 55.55f, 66.66f};
		System.out.println(f.length);

	}

}
```

--> In Java applications, if we want to read an element from an array then we have to use the following format.

```
    arrayRefVar[index]
```

EX:

```
package com.durgasoft.app02.test;

public class Test {

	public static void main(String[] args) {
		int[] a = {10,20,30,40,50};
		System.out.println(a[1]);//20
		System.out.println(a[3]);//40
		System.out.println(a[4]);//50
```

```
	}


}
```

--> While getting elements from an array on the basis of index values, if we provide an index value which is in out side range of the array indexes then JVM will raise an exception like "java.lang.ArrayIndexOutOfBoundsException"

EX:

```
package com.durgasoft.app02.test;


public class Test {


	public static void main(String[] args) {
		int[] a = {10,20,30,40,50};
		System.out.println(a[1]);//20
		System.out.println(a[3]);//40
		System.out.println(a[4]);//50
		System.out.println(a[10]);//ArrayIndexOutOfBoundsException
	}


}
```

EX:

```
package com.durgasoft.app02.test;


public class Test {


	public static void main(String[] args) {
```

```
                int[] a = {10,20,30,40,50};
                System.out.println(a[1]);
                System.out.println(a[2]);
                System.out.println(a[4]);
                System.out.println(a[a.length-2]);//a[5-2] =>a[3] => 40
                System.out.println(a[a.length]);//a[5]==>AIOBE
        }

}
```

--> In Declaare then initialize approach, size of the array is mandatory.

EX: int[] a = new int[];

----> Compilation Error

EX: int[] a = new int[5];

--> No Compilation Error, No Exception

EX: int[] a = new int[0];

---> No Compilation Error, if we add an element then we will get "java.lang.ArrayIndexOutOfBoundsException".

EX: int[] a = new int[-5];

---> No Compilation Error, java.lang.NegativeArraySizeException:

--> In Declare then intialize approach, we are able to add the elements upto the max index value, that is, size-1 , if we are trying to add an element at an index value which is in out side range of array indexes then JVM will raise an exception like "java.lang.ArrayIndexOutOfBoundsException".

EX:

```
package com.durgasoft.app02.test;

public class Test {

	public static void main(String[] args) {
		int[] a = new int[5];
		a[0] = 10; --> Valid
		a[1] = 20; --> Valid
		a[2] = 30; --> Valid
		a[3] = 40; --> Valid
		a[4] = 50; --> Valid
		a[5] = 60; --> Invalid,
java.lang.ArrayIndexOutOfBoundsException

	}
}
```

EX:

```
package com.durgasoft.app02.test;

public class Test {

	public static void main(String[] args) {
		int[] a = new int[5];
		a[0] = 10;
		a[1] = 20;
		a[2] = 30;
		a[3] = 40;
		a[4] = 50;
```

```
        //a[5] = 60; -> AIOBE
        System.out.println(a[1]);
        System.out.println(a[3]);
        System.out.println(a[4]);
        System.out.println(a[a.length-3]);//a[5-3]=>a[2] => 30
        //System.out.println(a[10]);---> AIOBE
        System.out.println(a[a.length]);//a[5] ==> AIOBE
    }
}
```

If we want to read all elements from an array then we have to use the following two approaches.

1. By Using refVar[index] Multiple times.
2. By Using Loops

1. By Using refVar[index] Multiple times:

```
package com.durgasoft.app02.test;

public class Test {

    public static void main(String[] args) {
        int[] a = {10,20,30,40,50};
        System.out.println(a[0]);
        System.out.println(a[1]);
        System.out.println(a[2]);
        System.out.println(a[3]);
        System.out.println(a[4]);
    }
```

}

The above approach is usefull when less no of elements are existed in an array, if we have more no of elements in an array then the above approach is not suggestible, in this case, we have to use loops.

2. By Using loops:

In Java, we are able to use three types of loops.

1. for Loop

2. while Loop

3. do-while Loop

By using all the above loops we are able to read elements from array, but, always, it is suggestible to use for loop, because, in java applications we will prefer to use for loop when we aware no of iterations over loop body in advance before writing loop.

In the case of arrays, if we want to read elements from an array by using loops then the no of iterations is same as the no of elements which are existed inside the array.

No Of Iterations in Loop = Size of the Array.

EX:

```
package com.durgasoft.app02.test;

public class Test {
	public static void main(String[] args) {
		int[] a = {10,20,30,40,50};
		for(int index = 0; index < a.length; index++) {
			System.out.println(a[index]);
```

```
		}
	}
}
```

EX:

```
package com.durgasoft.app02.test;

public class Test {

	public static void main(String[] args) {
		int[] a = {10,20,30,40,50};
		int index = 0;
		while(index < a.length) {
			System.out.println(a[index]);
			index = index + 1;
		}

	}

}
```

EX:

```
package com.durgasoft.app02.test;

public class Test {

	public static void main(String[] args) {
		String[] empNames = {"AAA", "BBB", "CCC", "DDD", "EEE"};
```

```
            int index = 0;
            do {
                System.out.println(empNames[index]);
                index = index + 1;
            } while (index < empNames.length);

        }

}
```

To read elements from an array if we use for loop then we are able to get the following problems.

```
int[] a = {10,20,30,40,50};
for(int index = 0; index < a.length; index++){
        System.out.println(a[index]);
}
```

1. We have to manage a seperate loop variable for looping purpose, it will wast the memory.
2. At each and every iteration JVM has to execute conditional expression, it will reduce java application performance, because, Consitional expressions are more streangthful operations, they may take more memory and more execution time.
3. At each and every iteration , JVM has to perform increment / decrement operation over the loop variable, it extra burdon for the JVM.
4. In this approach, we are getting array elements by providing index values explicitly, if the provided index values are proper then we will not get any problem, if the provided index values are not proper there there may be a chance to get an exception like "java.lang.ArrayIndexOutOfBoundsException".

To overcome all the above problems we have to use 'for-Each' loop provided by JAVA in its JDK1.5 version.

Syntax:

```
for(ArrayDataType var: ArrayRefVar){
    -----
}
```

EX:

```
int[] a = {10,20,30,40,50};
for(int element: a){
        System.out.println(element);
}
```

1. No Loop variable seperatly.
2. At each and every iteration, No conditionl expression is executed.
3. No increment / decrement operations are required.
4. We are not getting array elements on the basis of the array indexes, here we are getting elements directly from array by JVM.

EX:

```
package com.durgasoft.app02.test;

public class Test {

        public static void main(String[] args) {
                int[] a = {10, 20, 30, 40, 50};
                for(int element: a) {
                        System.out.println(element);
                }
                System.out.println();
```

```
		String[] empNames = new String[5];
		empNames[0] = "AAA";
		empNames[1] = "BBB";
		empNames[2] = "CCC";
		empNames[3] = "DDD";
		empNames[4] = "EEE";
		for(String empName: empNames) {
			System.out.println(empName);
		}
	}

}
```

Note: 'for-Each' loop is applicable for only retrieving elements from arrays and from Collections, 'for-Each' loop is not applicable for normal iterations, where we have to use normal for loop only.

2. Multi Dimensional Arrays:

--> It able to represent all the elements in more than one Dimension or in more than one level .

There are two ways to utilize multi dimensional arrays.

1. Declare and Initialize
2. Declare then Intialize

1. Declare and Initialize:

It able to declare the array and initialize the array with in single Statement.

Syntax:

DataType[][]...[] refVar = {{{{...},...},....},...};

EX: int[][] a = {{1,2,3},{2,3,4},{3,4,5},{4,5,6}};

2. Declare then Intialize:

It able to declare the array in one instruction and Initialize the array in another instructions.

Syntax:

DataType[][]...[] refVar = new DataType[size_1][size_2]...[size_n];

refVar[0][0]....[0] = val1;

refVar[0][0]....[1] = val2;

---

refVar[0][0]....[size_n-1] = val100;

---

refVar[0][size_2-1]...[size_n-1] = Val1000;

---

refVar[size_1-1][size_2-1]...[size_n-1] = val10000;

EX:

---

int[][] a = new int[4][3];

a[0][0] = 1;

a[0][1] = 2;

a[0][2] = 3;

a[1][0] = 2;

a[1][1] = 3;

a[1][2] = 4;

```
a[2][0] = 3;
a[2][1] = 4;
a[2][2] = 5;

a[3][0] = 4;
a[3][1] = 5;
a[3][2] = 6;
```

EX:

```
package com.durgasoft.app02.test;

public class Test {
	public static void main(String[] args) {
		int[][] a = {{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
		System.out.println(a);//[[I@abc123
		System.out.println(a.length);//4
		System.out.println(a[0]);//[I@a111
		System.out.println(a[0].length);//3
		System.out.println(a[0][1]);//2
	}
}
```

If we want to read elements from an array then we have to use the following two approaches.

1. By Using refVar[index1][index2]....[index_n]
2. By Using Loops

1. By Using refVar[index1][index2]....[index_n]:

EX:

```
package com.durgasoft.app02.test;

public class Test {
	public static void main(String[] args) {
		int[][] a = {{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
		System.out.println(a[0][0]);
		System.out.println(a[0][1]);
		System.out.println(a[0][2]);
		System.out.println();

		System.out.println(a[1][0]);
		System.out.println(a[1][1]);
		System.out.println(a[1][2]);
		System.out.println();

		System.out.println(a[2][0]);
		System.out.println(a[2][1]);
		System.out.println(a[2][2]);
		System.out.println();

		System.out.println(a[3][0]);
		System.out.println(a[3][1]);
		System.out.println(a[3][2]);
	}
}
```

2. By Using Loops

In Java , we are able to use three types of loops .

1. for Loop

2. while Loop

3. do-while Loop

EX:

```
package com.durgasoft.app02.test;


public class Test {
	public static void main(String[] args) {
		int[][] a = {{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
		for(int row = 0; row < a.length; row++) {
			for(int col = 0; col < a[row].length; col++) {
				System.out.print(a[row][col]+" ");
			}
			System.out.println();
		}
		System.out.println();
		int row = 0;
		while(row < a.length) {
			int col = 0;
			while(col < a[row].length) {
				System.out.print(a[row][col]+" ");
				col = col + 1;
			}
			row = row + 1;
			System.out.println();
```

```
            }
            System.out.println();

            int rowIndex = 0;
            do {
                int colIndex = 0;
                do {
                    System.out.print(a[rowIndex][colIndex]+" ");
                    colIndex = colIndex + 1;
                } while (colIndex < a[rowIndex].length);
                rowIndex = rowIndex + 1;
                System.out.println();
            } while (rowIndex < a.length);

        }
}
```

In Java applications, we are able to use for loop, while loop and do-while loop to retrieve elements from arrays, but, it is suggestible to use for loop as we know the no of iterations in advance before writing loop that is length of the array.

```
int[][] a = {{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
for(int row = 0; row < a.length; row++) {
    for(int col = 0; col < a[row].length; col++) {
        System.out.print(a[row][col]+" ");
    }
    System.out.println();
}
```

While retrieving elemnts from arrays , we are able to use for loop, but, it is suggestible to use for-Each loop.

```
for(ArrayDataType element: arrayRefVar){
        ----instructions----
}
```

EX:

```
package com.durgasoft.app02.test;

public class Test {
        public static void main(String[] args) {
                int[][] a = {{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
                for(int[] row: a) {
                        for(int val: row) {
                                System.out.print(val+" ");
                        }
                        System.out.println();
                }
        }
}
```

Q)Find the valid syntaxes from the following array declarations?

---

1) int[] a = new int[]; --> Invalid, Size is mandatory in SDA.

2) int[] a = new int[3];--> Valid

3) int[] a = new [3]int;--> Invalid

4) int[] a = new int[-5];--> No Errors, NegativeArraySizeException

5) int[] a = new int[2][3];--> Invalid

6) int[][] a = new int[3][3];-->Valid

7) int[][] a = new int[3][];--> Valid, Last dimension size is opt.

Note: In the above case, whgen we assign any value, then JVM will provide java.lang.NullPointerException.

8) int[][] a = new int[][3];--> Invalid

9) int[][] a = new int[][]; ---> Invalid

10)int a[][] = new int[3][3]; ---> Valid

11)int[] a[] = new int[3][3]; ---> Valid

12)int [][]a = new int[3][3]; ---> Valid

13)int []a[] = new int[3][3]; ---> Valid

In Java applications, we are able to declare arrays for both primitive data types and for User defined data types also like classes, abstract classes, interfaces, enums,.....

EX:

```
package com.durgasoft.app02.test;

class Course{
	String cid;
	String cname;
	int ccost;

	Course(String cid, String cname, int ccost){
		this.cid = cid;
		this.cname = cname;
		this.ccost = ccost;
	}
}

class Student{
	String sid;
	String sname;
```

```
        String saddr;
        Course[] courses;

        Student(String sid, String sname, String saddr, Course[] courses){
                this.sid = sid;
                this.sname = sname;
                this.saddr = saddr;
                this.courses = courses;
        }

        public void getStudentDetails() {
                System.out.println("Student Details");
                System.out.println("----------------------");
                System.out.println("Student Id       : "+sid);
                System.out.println("Student Name     : "+sname);
                System.out.println("Student Address  : "+saddr);
                System.out.println();
                System.out.println("CID\tCNAME\tCCOST");
                System.out.println("-----------------------");
                for(Course course: courses) {
                        System.out.print(course.cid+"\t");
                        System.out.print(course.cname+"\t");
                        System.out.print(course.ccost+"\n");
                }
        }
}
public class Test {
        public static void main(String[] args) {
                Course course1 = new Course("C-111", "C", 1000);
                Course course2 = new Course("C-222", "C++", 2000);
```

```
                Course course3 = new Course("C-333", "JAVA", 5000);
                Course course4 = new Course("C-444", ".NET", 4000);
                Course[] courses = {course1, course2, course3, course4};

                Student std = new Student("S-111", "Durga", "Hyd", courses);
                std.getStudentDetails();
        }
}
```

Anonymous Arrays:

--> Nameless Arrays are called as Anonymous Arrays.

--> We are able to utilize Anonymous arrays when we want to pass arrays as parameter to a method.

Syntax:

```
DataType[] refVar = new DataType[]{val1, val2,...val_n};

void m1(int[] a){
        ----
}

m1(new int[]{1,2,3,4});
```

EX:

```
package com.durgasoft.app02.test;
class Bank{
        public void displayCustomersNames(String[] customerNames) {
                for(String customerName: customerNames) {
                        System.out.println(customerName);
```

```
                }
        }
}
public class Test {
        public static void main(String[] args) {


                Bank bank = new Bank();
                bank.displayCustomersNames(new String[]{"AAA", "BBB",
"CCC", "DDD", "EEE"});
        }
}
```