Arrays:
—--------
Array is an object, it is able to store the same type of elements as per indexing, where the index value will start from 0 and it will end with size-1.

In Java, there are two types of Arrays.
1. Single Dimensional Arrays
2. Multi Dimensional Arrays

Single Dimensional Arrays:
—-------------------------
It is able to represent all the elements in a row.

There are two ways to use single dimensional Arrays.
1. Declare and Initialize
2. Declare then Initialize

Declare and Initialize:
In this approach we will declare the array and we will initialize the array in a single statement.

Syntax:
DataType[] refVar = {val1, val2,....,val_n};
EX: int[] a = {10,20,30,40,50};

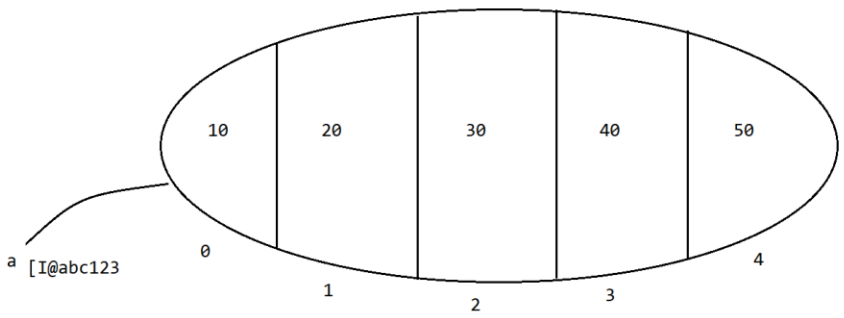
Declare then Initialize:
In this approach, we will declare the array in one statement and we will initialize the array in another statement.

Syntax:
```
DataType[] refVar = new DataType[size];
refVar[0] = val1;
refVar[1] = val2;
—--
—---
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;
```

```
int[] a = {10,20,30,40,50};
        or
int[] a = new int[5];
a[0] = 10;
a[1] = 20;
a[2] = 30;
a[3] = 40;
a[4] = 50;
```



In Java applications, an array is an object and its reference value is provided by toString() method internally which is provided by a class representing the array, here the toString() method is able to return a string contains the following value.

[[...Letter@refVal

Number of '[' representing the number of dimensions for the array.
Letter representing ArrayDataType
EX: int[] –> [I@abc123
EX: float[][] –-> [[F@abc123
EX: long[][][] –> [[[L@abc123

After declaring and initializing the array in the above approaches, we have to access the data from the array by using the following syntax.

DataType var = refVar[index];
EX: int val1 = a[0];

In the above syntax, if we provide the index value which is in the outside range of the array indexes then JVM will raise an exception like java.lang.ArrayIndexOutOfBoundsException.

```
int[] a = {10,20,30,40,50};
Sopln(a[0]);
Sopln(a[4]);
Sopln(a[10]); –> java.lang.ArrayIndexOutOfBoundsException
```

To read all elements from the array in the above approach we have to use the following instructions

```
int[] a = {10,20,30,40,50};
Sopln(a[0]);
Sopln(a[1]);
Sopln(a[2]);
Sopln(a[3]);
Sopln(a[4]);
```

In the above case we are writing duplicate code every time, here to reduce duplicate code we have to use loops.
If we want to read elements from the array then we have to use a for loop.

Note: In Arrays, there is a variable ‘length’ representing array size.

EX:

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

```
Index:     OP
 0         10
 1         20
 2         30
 3         40
 4         50
```

To read elements from the array if we use the above approach then we are able to get the following problems.

1. We have to manage a separate variable for looping purposes.
2. We have to execute a conditional expression at each and every iteration, it will take more memory and more execution time, because conditional expressions are more strengthful expressions.
3. We have to perform increment / decrement operations over a loop variable at each and every iteration.
4. We are providing index values explicitly to the array inorder to get array elements, here if the index value is not proper then there is a chance to get ArrayIndexOutOfBoundsException.

All the above problems are able to reduce java applications performance.

To overcome all the above problems, JDK1.5 version has provided an enhancement in the for loop that is for-Each loop.

Syntax:

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

In the case of for-Each loop, JVM is able to read element by element from the array and assign element by element to the variable in the forEach loop and it will execute the loop body for each and every element.

EX:

```
int[] a = {10,20,30,40,50};
for(int x: a){
      Sopln(x);
}
```

EX:

```
public class Main {
public static void main(String[] args){
//int[] a = {10,20,30,40,50};
int[] a = new int[5];
a[0] = 10;
a[1] = 20;
a[2] = 30;
a[3] = 40;
a[4] = 50;
System.out.println(a);// [I@abc123
System.out.println(a.length);// 5
System.out.println(a[2]);// 30
System.out.println(a[4]);// 50
//System.out.println(a[5]);ArrayIndexOutOfBoundsExc
eption
//System.out.println(a[a.length]);
ArrayIndexOutOfBoundsException
```

```
//System.out.println(a[a.length-6]);
ArrayIndexOutOfBoundsException
System.out.println(a[a.length-2]);// 40
System.out.println();
for(int index = 0; index < a.length; index++){
System.out.println(a[index]);
}
System.out.println();
for (int element: a){
System.out.println(element);
}
}
}
```

```
[I@65ab7765
5
30
50
40

10
20
30
40
50

10
20
30
40
50
```

Multi Dimensional Arrays:
—------------------------
Multi Dimensional Arrays are able to represent data in more than one dimension or level.

To use Multi Dimensional Arrays in java applications we have to use the following approaches.

1. Declare and Initialize
2. Declare then Initialize

Declare and Initialize:
In this approach, we will declare the array and we will initialize the array in a single statement.

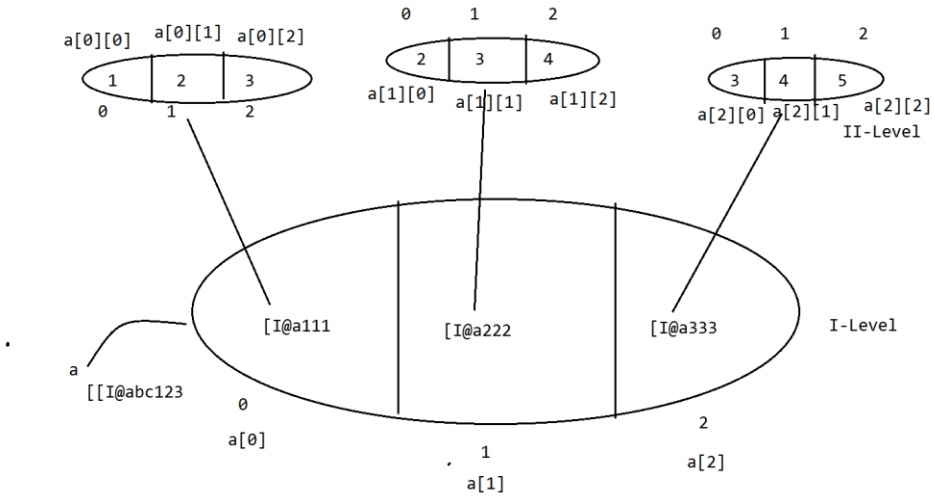
Syntax:
```
DataType[][][]...[] refVar = {{{...{val1,val2,...val-n},...}...},...};
```
EX:
```
int[][] a = {{1,2,3},{2,3,4},{3,4,5}};
Int[][][] a = {{{1,2,3},{2,3,4}},{{3,4,5},{4,5,6}},{{5,6,7},{6,7,8}}}
```

Declare then Initialize :
In this approach, we will declare the array in one statement and we will initialize the array in another statements.
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] = val_x;
—-
—-
refVar[0][size_2-1]....[size_n-1] = val_y;
—-
—-
refVar[size_1-1][size_2-1]...[size_n-1] = val_n;
```

EX:
```
int[][] a = new int[3][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;
```

```
int[][] a = {{1,2,3},{2,3,4},{3,4,5}};
    or        0     1     2
int[][] a = new int[3][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;
```



EX:

```
public class Main {
public static void main(String[] args) {
int[][] a = {{1,2,3},{2,3,4},{3,4,5}};
System.out.println(a);
System.out.println(a.length);
System.out.println(a[0]);
System.out.println(a[2]);
//System.out.println(a[3]); -->
ArrayIndexOutOfBoundsException
System.out.println(a[1].length);
System.out.println(a[1][1]);
System.out.println(a[2][1]);
//System.out.println(a[1][3]);-->
ArrayindexOutOfBoundsException
System.out.println(a[a.length-1][a.length-2]);//
a[2][1]
```

```
System.out.println(a[a[2].length-1][a[1].length-
2]);//a[2][1]
System.out.println();

for(int row = 0; row < a.length; row++){
for(int col = 0;col < a[row].length; col++ ){
System.out.print(a[row][col]+"\t");
}
System.out.println();
}
System.out.println();
for(int[] row: a){
for(int val: row){
System.out.print(val+"\t");
}
System.out.println();
}



}
}
```

```
[[I@65ab7765
3
[I@1b28cdfa
[I@eed1f14
3
3
4
4
4

1	2	3
2	3	4
3	4	5
```

```
1     2     3
2     3     4
3     4     5
```

In Java applications, we are able to declare arrays for the user defined data types like Classes, abstract classes, interfaces….

EX:
Course.java

```java
package com.durgasoft.entities;

public class Course {
private String courseId;
private String courseName;
private int courseFee;

public Course(String courseId, String courseName, int
courseFee) {
this.courseId = courseId;
this.courseName = courseName;
this.courseFee = courseFee;
}

public String getCourseId() {
return courseId;
}

public String getCourseName() {
return courseName;
}

public int getCourseFee() {
return courseFee;
```

```
}
}
```

Student.java

```
package com.durgasoft.entities;

public class Student {
private String studentId;
private String studentName;
private String studentAddress;
private Course[] courses;

public Student(String studentId, String studentName, String studentAddress, Course[] courses) {
this.studentId = studentId;
this.studentName = studentName;
this.studentAddress = studentAddress;
this.courses = courses;
}
public void getStudentDetails(){
System.out.println("Student Details");
System.out.println("----------------------");
System.out.println("Student Id : "+studentId);
System.out.println("Student Name : "+studentName);
System.out.println("Student Address : "+studentAddress);
System.out.println("CID\t\tCNAME\tCFEE");
System.out.println("-----------------------");
for(Course course: courses){
System.out.print(course.getCourseId()+"\t");
System.out.print(course.getCourseName()+"\t");
System.out.print(course.getCourseFee()+"\n");
}
}
}
```

Main.java

```
import com.durgasoft.entities.Course;
import com.durgasoft.entities.Student;

public class Main {
public static void main(String[] args) {
Course course1 = new Course("C-111", "JAVA", 30000);
Course course2 = new Course("C-222", "PYTHON", 20000);
Course course3 = new Course("C-333", "DEVOPS", 10000);
Course[] courses = {course1, course2, course3};

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

```
}
```

Anonymous Arrays:
—----------------
Anonymous arrays are the nameless arrays, they will be used to pass arrays as parameters to the methods without declaring arrays.

Syntax:
new DataType[]{val1, val2, val3,....};

EX:
Bank.java

```
package com.durgasoft.entities;

public class Bank {
public void displayCustomerNames(String[]
customerNames){
for(String custName: customerNames){
System.out.println(custName);
}
}
}
```

Main.java

```
import com.durgasoft.entities.Bank;

public class Main {
public static void main(String[] args) {
Bank bank = new Bank();
//String[] custNames = {"AAA", "BBB", "CCC", "DDD",
"EEE"};
bank.displayCustomerNames(new String[]{"AAA", "BBB",
"CCC", "DDD", "EEE"});

}
```

```
}
```

Note: IN general, in java applications anonymous arrays are suitable for the less number of values inside the array, if we have more values inside the array it is suggestible to use explicit arrays.

Note: Anonymous arrays are used to test the java methods.

Q)Find the valid syntaxes from the following list of arrays declaration?
—----------

Ans:
—---

1. int[] a = {10,20,30,40}; —--> Valid
2. int[3] a = {10,20,30}; —----> Invalid
3. int a[] = {10,20,30,40};----> Valid

4. int[] a = new int[]; —------> Invalid
5. int[] a = new int[-5]; —----> No Error, but NegativeArraySizeException
6. int[] a = new int[5.4];-----> INvalid
7. int[] a = new int[0]; —-----> Valid

8. int[][] a = new int[5][4]; —---> Valid
9. int[][] a = new int[][4]; —----> INvalid
10. int[][] a = new int[5][]; —----> Valid

11. int[] a[] = new int[3][3];—----> Valid
12. int[] []a = new int[3][3];-----> Valid
13. []int []a = new int[3][3];-----> INvalid
14. int[] []a = new int[3]; —-----> Invalid