```
Why we need Arrays?
  To store mulitple values of same type in single varaible we need Arrays.

Note:
1. Arrays in java are treated as "Objects"
2. In java memory for objects is given in "HeapArea".
3. If memory is given in "HeapArea", jvm will give default value for the variables
based on the datatype.
4. default value for boolean variable is "false".
5. If memory is given in "StackArea", programmer should initialise the value before
using the variable, otherwise it would
   result in "CompileTimeError".
6. While working with Arrays in java, we should follow 3 steps
      a. Array Declaration
      b. Array Construction(new)
      c. Array Intialization
7. Any problem occured at runtime we categorise into 2 types
      a. Exception => Problem occured and it can be handled.
      b. Error     => Problem occured,but it can't be handled.


++++++++++++++++++++++++++++
Shortcut to work with Arrays
++++++++++++++++++++++++++++

Shortcut of way declartion,construction,initialisation in single line
=====================================================================
  int[]a = {10,20,30,40};
  char[] a= {'a','e','i','o','u'};
  String[] a= {"sachin","ramesh","tendulkar","IND"};


Array Element Assignements
==========================
case 1:: In case of primitive array as an array element any type is allowed which
can be promoted to declared type.

eg#1.
   int[] a=new int[10];
    a[0]=97;
    a[1]='a';
    byte b= 10;
    a[2]=b;
    short s=25;
     a[3]=s;
     a[4]=10L;//CE: possible loss of precession


case 2:: In case of Object type array as an array elements we can provide either
declared type object or its child class objects.
eg#1.
     Object[] obj=new Object[5];
     obj[0] =new Object();//valid
     obj[1] =new Integer(10);//valid
     obj[2] =new String("sachin");//valid

eg#2.
    Number[] num=new Number[10];
     num[0]=new Integer(10);//valid
     num[1]=new Double(10.0);//valid
     num[2]=new String("sachin");//invalid
```

case3:: In case of interface type array as an array element we can provide its implementation class Object.

```
      Runnable[] r=new Runnable[5];
       r[0]= new Thread("sachin");
       r[1]= new String("dhoni");//CE
```

case4:: In case of abstract class type array as an array element we can provide its child class object.

Array variable assignment
========================
case1:: Element level type promotion is not applicable
eg:: char value can be type promoted to int, but char[] can't be type promoted to int[].

```
int[] a= {1,2,3};
char[] c={'a','b','c'};
int[] b = a;
int[] a = c;//invalid
```

which of the following promotions are valid?

```
   char ---->  int      [valid]
   char[] -->  int[]    [invalid]
   int    -->  long     [valid]
   int[]  ==>  long[]   [invalid]
   double ==>  float    [valid]
   double[]==> float[]  [invalid]
+++++++++++++++++++++++++++++++++++++++
   String  ==> Object   [valid]
   String[] => Object[] [valid]
```

case2:: In case of Object type array,its child type array can be assigned.

```
          eg:: String[] names={"sachin","saurav","dhoni"};
               Object[] obj=names;
```

case3:: Whenever we are assigning one array reference to another array reference,its just
the reference which are being copied not the array elements.
While copying the reference only its type would be given importance,not its size.

```
eg:: int[] a= {10,20,30,40};
     int[] b= {100,200};
     a=b;
     b=a;
```

case4:: Whenever we are copying the array,its reference will be copied but we should match it with
the array dimension and its type,otherwise it would result in compile time error.

```
eg:: int[][] a= {{10},{20},{30}};
     int[] b={100,200,300};
         b= a; //CE: incompatible type
```

Note:: In array assignment,its type and dimension must be matched otherwise it

```
would result in compile time error.

       int[] a ={10,20,30};S.o.p(a);//[I@..
       float[] f={10.0f,20.0f}; S.o.p(f); //[F@..
       boolean[] b= {true,false}; S.o.p(b); //[Z@..
       Integer[] i={10,20,30}; S.o.p(i);//[L@...
       Float[] f = {10.0f,20.0f}; S.o.p(i); //[L@...


Working with 2D-Arrays
++++++++++++++++++++++
2D-Array =  1D-Array + 1D-Array
               (ref)       (data)

Declaration(All are valid)
    int[][] a ;//recomended
    int  a[][];
    int  [][]a;
    int[] []a;
    int[] a[];
    int []a[];

ArrayConstruction
    int[][] a =new int[3][2];
            or
    int[][] a= new int[3][];
      a[0]=new int[5];
      a[1]=new int[3];
      a[2]=new int[1];

ArrayInitalisation
   a[0][0] = 10;
   a[2][3] = 5;

Program to demonstrate 2-D array
++++++++++++++++++++++++++++++++
class Test
{
      public static void main(String[] args)
      {
            //Array Declaration
            int[][] a= null;

            //Array Construction
             a =new int[2][3];
             System.out.println(a);

            //Array initialization
            a[0][0] = 10;
            a[0][1] = 20;
            a[0][2] = 30;

            a[1][0] = 50;
            a[1][1] = 60;
            a[1][2] = 70;


             for (int i = 0;i<a.length ;i++ )//2-D
             {
                   for (int j= 0; j<a[i].length;j++ )//1-D
                   {
                         System.out.print(a[i][j] +"\t");
                   }
                   System.out.println();
             }
      }
}
```

```
Output
D:\OctBatchMicroservices>javac Test.java
D:\OctBatchMicroservices>java Test
[[I@76ed5528
10      20      30
50      60      70
```

Tricky Questions
================

```
  a. int[] a,b;     => a-1D, b-1D
  b. int a[],b[];   => a-1D, b-1D
  c. int a[],b;     => a-1D, b-variable
  d. int a,[]b;     => CE
  e. int []a,[]b;   => CE
  f. int []a,b;     => a-1D,b-1D
```

Which of the following declarations are valid?

```
  int[] a,b;    // a-1D, b-1D
  int[] a[],b;  // a-2D,  b-1D
  int[] []a,b;  // a-2D, b-2D
  int[] a, []b; //CE
```

Note: if we want to specify the dimension before the variable, that rule is applicable only for first varaible,second varaible onwards we can't apply in the same declaration.

Shortcut way to create 2-D array
++++++++++++++++++++++++++++++++

```
class Test
{
      public static void main(String[] args)
      {
            int[] a={10,20,30};
            System.out.println("1-D Array");
            for (int i=0;i<a.length ;i++ )
            {
                  System.out.print(a[i]+"\t");
            }

            System.out.println();
            System.out.println("*****************");

            int[][] b= {
                              {10,20,30,40},
                              {50,60}
                       };

            System.out.println("2D-Array");
            for (int i =0;i<b.length ;i++ )
            {
                  System.out.println("1D-Array");
                  for (int j=0;j<b[i].length ;j++ )
                  {
                        System.out.print(b[i][j] +"\t");
                  }
                  System.out.println();
            }

      }
}
```

Output

```
D:\OctBatchMicroservices>javac Test.java
D:\OctBatchMicroservices>java Test
1-D Array
10      20      30
*****************
2D-Array
1D-Array
10      20      30      40
1D-Array
50      60
```

Note: if an object does'nt have reference to access, then such objects are called as "Garbage" Object

Q>

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

Totally how many objects are created?
Ans. 11
How many objects are eligible for Garbage Collection?
Ans. 6

Collecting inputs from the User
++++++++++++++++++++++++++++++
 1. We use Scanner class which is present inside "java.util" folder
 2. To read integer inputs we use a method called "nextInt()".

eg#1.

```
import java.util.Scanner;

class Test
{
      public static void main(String[] args)
      {
            int size=0;

            System.out.print("Enter the size of the array:");
            Scanner scan= new Scanner(System.in);
            size = scan.nextInt();

            //Array Declaration and Construction
            int[] arr = new int[size];

            System.out.println(arr);
            System.out.println("The size of the array is :: "+arr.length);
            System.out.println("Before initialization");
            for (int i =0;i<arr.length ;i++ )
            {
                  System.out.print(arr[i]+"\t");
            }

            System.out.println();

            //Array initalization
            for (int i =0;i<arr.length ;i++ )
            {
                  System.out.print("Enter the array value ::  ");
                  arr[i]=scan.nextInt();
            }


            System.out.println("ONE-D Array elements are :: ");
            for (int i=0; i<arr.length; i++)
            {
                  System.out.print(arr[i]+"\t");
            }
      }
}
```

Output

```
D:\OctBatchMicroservices>javac Test.java
D:\OctBatchMicroservices>java Test
Enter the size of the array:5
[I@2d98a335
The size of the array is :: 5
Before initialization
0       0       0       0       0
Enter the array value ::  10
Enter the array value ::  20
Enter the array value ::  30
Enter the array value ::  40
Enter the array value ::  50
ONE-D Array elements are ::
10      20      30      40      50
```

+++++++++++++++++++++++++++
Working with foreach loops
+++++++++++++++++++++++++++

1. What is the need for foreach loop when we already have forloop?

Ans. To work with for loop as a programmer we need to do the following things

  a. initalization
  b. putting condition
  c. performing inc/decr depending on initalisation
  d. Access the element or put the element into array through index

Solution : foreach loop(just read the element, don't worry about initalization,condition and incr/dec)

```
         syntax:
            for(datatype variable:iterating-object)
            {
                  //Access the variable directly
            }
```

Program to demonstrate foreach loop
+++++++++++++++++++++++++++++++++++

```
import java.util.Scanner;

class Test
{
      public static void main(String[] args)
      {
            int size=0;

            System.out.print("Enter the size of the array:");
            Scanner scan= new Scanner(System.in);
            size = scan.nextInt();

            //Array Declaration and Construction
            int[] arr = new int[size];

            System.out.println(arr);
            System.out.println("The size of the array is :: "+arr.length);
            System.out.println("Before initialization");
            for(int data : arr)
            {
                  System.out.print(data + "\t");
            }

            System.out.println();

            //Array initalization
            for (int i =0;i<arr.length ;i++ )
            {
                  System.out.print("Enter the array value ::  ");
                  arr[i]=scan.nextInt();
            }

            System.out.println("ONE-D Array elements are :: ");
            for(int data:arr)
            {
                  System.out.print(data+"\t");
            }
            System.out.println();

      }
}
D:\OctBatchMicroservices>javac Test.java
D:\OctBatchMicroservices>java Test
Enter the size of the array:5
[I@2d98a335
The size of the array is :: 5
Before initialization
0       0       0       0       0
Enter the array value ::  10
Enter the array value ::  20
Enter the array value ::  30
Enter the array value ::  40
Enter the array value ::  50
ONE-D Array elements are ::
10      20      30      40      50
```

eg#2.

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

```
class Test
{
    public static void main(String[] args)
    {
        int[][] arr = {{10,20,30},{40,50},{60}};

        System.out.println("using forloop 2-D array");
        for (int i=0;i<arr.length ;i++ )
        {
            for (int j=0;j<arr[i].length ;j++ )
            {
                System.out.print(arr[i][j]+"\t");
            }
            System.out.println();
        }

        System.out.println("************************");
        System.out.println("using foreach loop 2-D array");
        for(int[] oneDArr:arr)
        {
            for ( int data: oneDArr )
            {
                System.out.print(data + "\t");
            }
            System.out.println();
        }
        System.out.println();

    }
}
```

2. What is the difference b/w for loop and foreach loop?

```
    forloop      => meant for both read and write purpose.
                => it might lead to AIOBE, if we dont' use it properly.
                => by placing the condition we can iterate from R to L and L to R.
                => Accessing of elements is through  index only.

    foreach loop => meant for only read purpose
                 The problem of exception won't occur.
                 iteration is possible only from L to R.
                 Accessing of elements is through "datatype" of the variable.
```

++++++++
Snippets
++++++++
Question

```
1. public class BuildStuff {
2.    public static void main(String[] args) {
3.          boolean test = true;
4.          Integer x = 343;
5.          Integer y = new BuildStuff().go(test, x);
6.          System.out.println(y);
7.    }
8.    int go(boolean b, int i) {
9.          if(b) return (i/7);// return 343/7  -> return 49
10.         return (i/49);
11.   }
12.}
```

What is the result?
A. 7
B. 49// Answer
C. 343
D. Compilation fails.
E. An exception is thrown at runtime.

Q>
Given:

```
public class TestString1 {
    public static void main(String[] args) {
          String str = "420";
          str += 42; // str = str+42  = 42042
          System.out.print(str);
    }
}
```

What is the output?
A. 42
B. 420
C. 462
D. 42042//Answer
E. Compilation fails.
F. An exception is thrown at runtime.

Q>
Given:

```
public class Pass2 {
      public void main(String [] args) {
            int x = 6;
            Pass2 p = new Pass2();
            p.doStuff(x);
            System.out.print(" main x = " + x);
      }
      void doStuff(int x) {
            System.out.print(" doStuff x = " + x++);
      }
}
```

And the command-line invocations:

```
javac Pass2.java
java Pass2
```

What is the result?
A. Compilation fails.
B. An exception is thrown at runtime.
C. doStuff x = 6 main x = 6 //Answer
D. doStuff x = 6 main x = 7
E. doStuff x = 7 main x = 6
F. doStuff x = 7 main x = 7