Array

An array is collection of similar type of data.

or

An array is set of homogeneous data elements.

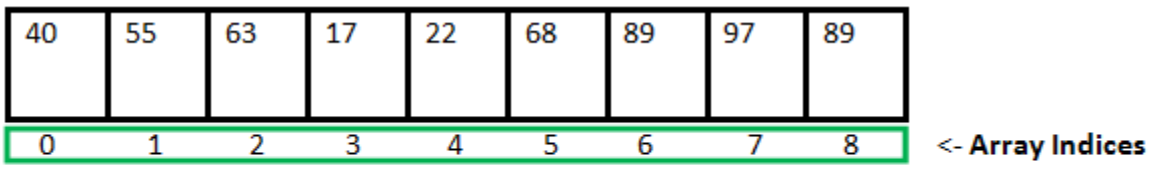
or

An array is a group of like-typed variables that are referred to by a common name.

Following are some important point about Java arrays.

- In Java all arrays are dynamically allocated.
- Since arrays are objects in Java, we can find their length using member length.
- A Java array variable can also be declared like other variables with [] after the data type.
- The variables in the array are ordered and each have an index beginning from 0.
- Java array can be also be used as a static field, a local variable or a method parameter.
- The **size** of an array must be specified by an int value and not long or short.
- The direct superclass of an array type is Object.
- Every array type implements the interfaces Cloneable and java.io.Serializable.

Array can contains primitives data types as well as objects of a class depending on the definition of array. In case of primitives data types, the actual values are stored in contiguous memory locations. In case of objects of a class, the actual objects are stored in heap segment.



Array Length = 9
First Index = 0
Last Index = 8

Creating, Initializing, and Accessing an Array

One-Dimensional Arrays :

The general form of a one-dimensional array declaration is

datatype varname[];
OR
datatype []varname;

An array declaration has two components: the type and the name. *type* declares the element type of the array. The element type determines the data type of each element that comprises the array. Like array of int type, we can also create an array of other primitive data types like char, float, double etc. or user defined data type(objects of a class).Thus, the element type for the array determines what type of data the array will hold.

Example:

```
// both are valid declarations
int Array[];
or int[] Array;
// an array of references to objects of
// the class MyClass (a class created by
// user)
MyClass myClassArray[];

Object[]  ao,      // array of Object
Collection[] ca;  // array of Collection
                  // of unknown type
```

Although the above first declaration establishes the fact that int Array is an array variable, **no array actually exists**. It simply tells to the compiler that this(int Array) variable will hold an array of the integer type. To link int Array with an actual, physical array of integers, you must allocate one using **new** and assign it to int Array.

Instantiating an Array in Java

When an array is declared, only a reference of array is created. To actually create or give memory to array, you create an array like this: The general form of *new* as it applies to one-dimensional arrays appears as follows:

```
varname = new datatype [length];
```

Here, *type* specifies the type of data being allocated, *length* specifies the number of elements in the array, and *var-name* is the name of array variable that is linked to the array. That is, to use *new* to allocate an array, **you must specify the type and number of elements to allocate.**

Example:

```
int Array[];    //declaring array
Array = new int[20];  // allocating memory to array
```

OR

```
int[] Array = new int[20]; // combining both statements in one
```

Note :

1. The elements in the array allocated by *new* will automatically be initialized to **zero**(for numeric types), **false** (for boolean), or **null** (for reference types).Refer Default array values in Java
2. Obtaining an array is a two-step process. First, you must declare a variable of the desired array type. Second, you must allocate the memory that will hold the array, using new, and assign it to the array variable. Thus, **in Java all arrays are dynamically allocated.**

Accessing Java Array Elements using for Loop

Each element in the array is accessed via its index. The index begins with 0 and ends at (total array size)-1. All the elements of array can be accessed using Java for Loop.

```
// accessing the elements of the specified array
for (int i = 0; i < arr.length; i++)
  System.out.println("Element at index " + i +" : "+ arr[i]);

//length is variable to find length of any array variable
```

Compile time initialization of Array

```
int arr[] =new int[5];
arr[0] = 10;
arr[1] = 20;
arr[2] = 30;
arr[3] = 40;
 arr[4] = 50 ;
```

Array Literal

In a situation, where the size of the array and variables of array are already known, array literals can be used.

```
int[] Array = {1,2,3,4,5,6,7,8,9,10 };
// Declaring array literal
```

The length of this array determines the length of the created array.

There is no need to write the new int[] part in the latest versions of Java

Run time initialization of Array

```
int arr[] =new int[5];
Scanner sn=new Scanner (System.in);
arr[0]=sn.nextInt();
arr[1]=sn.nextInt();
arr[2]=sn.nextInt();
arr[3]=sn.nextInt();
arr[4]=sn.nextInt();
```

We can also use loop to minimize the lines of code like:

```
int arr[] =new int[5];
Scanner sn=new Scanner (System.in);
for(i=0;i<=4;i++)
arr[i]=sn.nextInt();
```

Implementation:

```
// Java program to illustrate creating an array
// of integers,  puts some values in the array,
// and prints each value to standard output.

class ABC
{
   public static void main (String[] args)
   {
     // declares an Array of integers.
     int[] arr;

     // allocating memory for 5 integers.
     arr = new int[5];

     // initialize the first elements of the array
     arr[0] = 10;

     // initialize the second elements of the array
     arr[1] = 20;

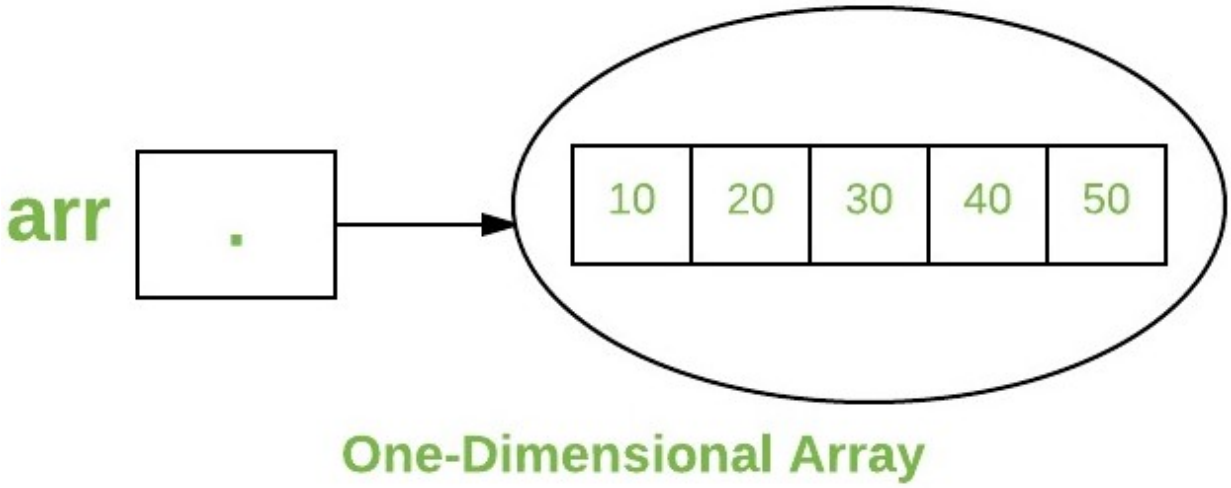
     //so on...
     arr[2] = 30;
     arr[3] = 40;
     arr[4] = 50;

     // accessing the elements of the specified array
     for (int i = 0; i < arr.length; i++)
       System.out.println("Element at index " + i +
                                " : "+ arr[i]);
   }
}
```

Output:

```
Element at index 0 : 10
```

```
Element at index 1 : 20
Element at index 2 : 30
Element at index 3 : 40
Element at index 4 : 50
```



Arrays of Objects

An array of objects is created just like an array of primitive type data items in the following way.

```
Student[] arr = new Student[7]; //student is a user-defined class
```

The studentArray contains seven memory spaces each of size of student class in which the address of seven Student objects can be stored. The Student objects have to be instantiated using the constructor of the Student class and their references should be assigned to the array elements in the following way.

```
Student[] arr = new Student[5];
```

```
// Java program to illustrate creating an array of
// objects


class Student
{
    public int roll_no;
```

```
    public String name;
    Student(int roll_no, String name)
    {
        this.roll_no = roll_no;
        this.name = name;
    }
}

// Elements of array are objects of a class Student.
public class ABC
{
    public static void main (String[] args)
    {
        // declares an Array of integers.
        Student[] arr;

        // allocating memory for 5 objects of type Student.
        arr = new Student[5];

        // initialize the first elements of the array
        arr[0] = new Student(1,"aman");

        // initialize the second elements of the array
        arr[1] = new Student(2,"vaibhav");

        // so on...
        arr[2] = new Student(3,"shikar");
        arr[3] = new Student(4,"dharmesh");
        arr[4] = new Student(5,"mohit");

        // accessing the elements of the specified array
        for (int i = 0; i < arr.length; i++)
            System.out.println("Element at " + i + " : " +
                    arr[i].roll_no +" "+ arr[i].name);
    }
```

```
}
```

Output:

```
Element at 0 : 1 aman
Element at 1 : 2 vaibhav
Element at 2 : 3 shikar
Element at 3 : 4 dharmesh
Element at 4 : 5 mohit
```

What happens if we try to access element outside the array size?

Compiler throws **ArrayIndexOutOfBoundsException** to indicate that array has been accessed with an illegal index. The index is either negative or greater than or equal to size of array.

```
class Example
{
   public static void main (String[] args)
   {
      int[] arr = new int[2];
      arr[0] = 10;
      arr[1] = 20;


      for (int i = 0; i <= arr.length; i++)
         System.out.println(arr[i]);
   }
}
```

Runtime error

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 2
   at Example.main(File.java:12)
```

Output:

```
10
20
```

Multidimensional Arrays

Multidimensional arrays are **arrays of arrays** with each element of the array holding the reference of other array. These are also known as Jagged Arrays. A multidimensional array is created by appending one set of square brackets ([]) per dimension.

Syntax:

```
data_type[1st dimension][2nd dimension][]..[Nth dimension] array_name
 = new data_type[size1][size2]....[sizeN];
```

where:

- **data_type**: Type of data to be stored in the array. For example: int, char, etc.
- **dimension**: The dimension of the array created.
  For example: 1D, 2D, etc.
- **array_name**: Name of the array
- **size1, size2, …, sizeN**: Sizes of the dimensions respectively.

Examples:

```
int[][] intArray = new int[10][20]; //a 2D array or matrix
int[][][] intArray = new int[10][20][10]; //a 3D array
```

Size of multidimensional arrays: The total number of elements that can be stored in a multidimensional array can be calculated by multiplying the size of all the dimensions.

For example:

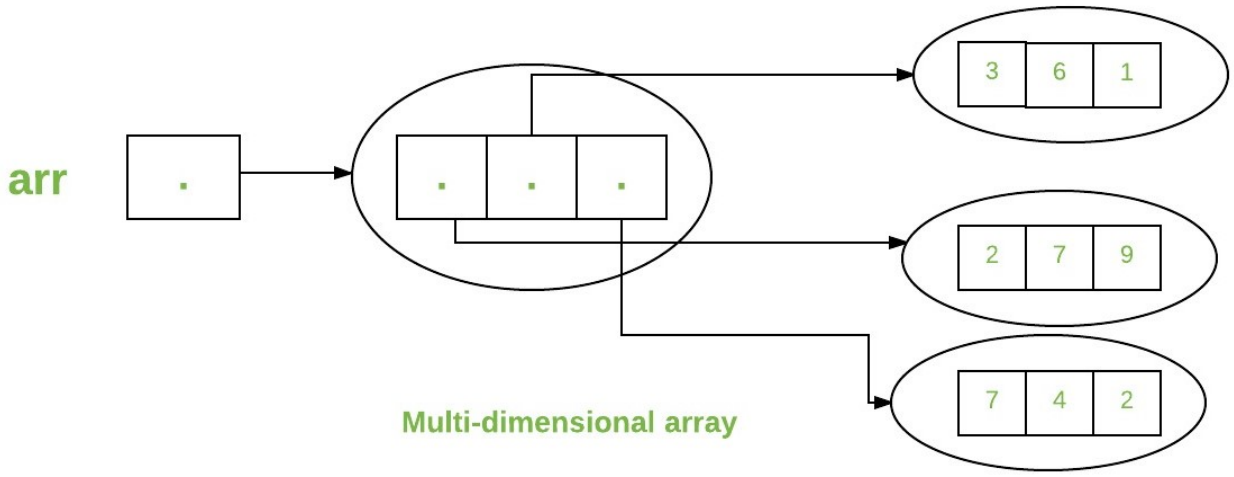
The array **int[][] x = new int[10][20]** can store a total of (10*20) = 200 elements.
Similarly, array **int[][][] x = new int[5][10][20]** can store a total of (5*10*20) = 1000 elements.

```
class multiDimensional
{
   public static void main(String args[])
   {
      // declaring and initializing 2D array
      int arr[][] = { {2,7,9},{3,6,1},{7,4,2} };
        // printing 2D array
      for (int i=0; i< 3 ; i++)
      {
         for (int j=0; j < 3 ; j++)
            System.out.print(arr[i][j] + " ");
          System.out.println();
```

```
    }
  }
}
```

Output:

```
2 7 9
3 6 1
7 4 2
```



Two – dimensional Array (2D-Array)

Two – dimensional array is the simplest form of a multidimensional array. A two – dimensional array can be seen as an array of one – dimensional array for easier

understanding.

Indirect Method of Declaration:

- **Declaration – Syntax:**
- **data_type[][] array_name = new data_type[row_length][column_length];**
- For example: int[][] arr = new int[10][20];
- **Initialization – Syntax:**
- **array_name[row_index][column_index] = value;**
- For example: arr[0][0] = 1;

Example:

```
class Example {
   public static void main(String[] args)
   {
       int[][] arr = new int[10][20];
      arr[0][0] = 1;
       System.out.println("arr[0][0] = " + arr[0][0]);
   }
}
```

Output:

```
arr[0][0] = 1;
```

Direct Method of Declaration:
Syntax:

```
data_type[][] array_name = {
                    {valueR1C1, valueR1C2, ....},
                    {valueR2C1, valueR2C2, ....}
                  };
```

For example: int[][] arr = {{1, 2}, {3, 4}};

Example:

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


      int[][] arr = { { 1, 2 }, { 3, 4 } };


      for (int i = 0; i < 2; i++)
         for (int j = 0; j < 2; j++)
            System.out.println("arr[" + i + "][" + j + "] = "
```

```
                    + arr[i][j]);
    }
}
```

Output:

```
arr[0][0] = 1
arr[0][1] = 2
arr[1][0] = 3
arr[1][1] = 4
```

Accessing Elements of Two-Dimensional Arrays

Elements in two-dimensional arrays are commonly referred by **x[i][j]** where 'i' is the row number and 'j' is the column number.

Syntax:

```
x[row_index][column_index]
```

For example:

```
int[][] arr = new int[10][20];
arr[0][0] = 1;
```

The above example represents the element present in first row and first column.

Note: In arrays if size of array is N. Its index will be from 0 to N-1. Therefore, for row_index 2, actual row number is 2+1 = 3.

Example:

```
class ABC {
    public static void main(String[] args)
    {
        int[][] arr = { { 1, 2 }, { 3, 4 } };
        System.out.println("arr[0][0] = " + arr[0][0]);
    }
}
```

Output:

```
arr[0][0] = 1
```

Compile time initialization of Array

```
int arr[][] =new int[2][3];
arr[0][0] = 10;
```

```
        arr[0][1] = 20;
        arr[0][2] = 30;
        arr[1][0] = 40;
arr[1][1] = 50;
arr[1][2] = 60;
```

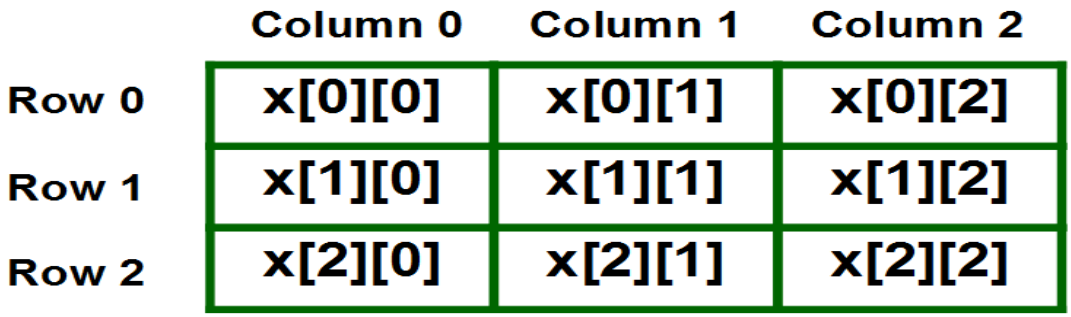
Run time initialization of Array

```
int arr[][] =new int[2][3];
Scanner sn=new Scanner (System.in);
arr[0][0]=sn.nextInt();
arr[0][1]=sn.nextInt();
arr[0][2]=sn.nextInt();
arr[1][0]=sn.nextInt();
arr[1][1]=sn.nextInt();
arr[1][2]=sn.nextInt();
```

We can also use loop to minimize the lines of code like:

```
int arr[][] =new int[2][3];
Scanner sn=new Scanner (System.in);
for(i=0;i<=1;i++)
{
        for(j=0;j<=2;j++)
        {
                arr[i][j]=sn.nextInt();
        }
}
```

Representation of 2D array in Tabular Format: A two – dimensional array can be seen as a table with 'x' rows and 'y' columns where the row number ranges from 0 to (x-1) and column number ranges from 0 to (y-1). A two – dimensional array 'x' with 3 rows and 3 columns is shown below:



Print 2D array in tabular format:

To output all the elements of a Two-Dimensional array, use nested for loops. For this two for loops are required, One to traverse the rows and another to traverse columns.

Example:

```
class ABC {
   public static void main(String[] args)
   {
       int[][] arr = { { 1, 2 }, { 3, 4 } };
       for (int i = 0; i < arr.length-1; i++)  //arr.length for row
      {
     for (int j = 0; j < arr[0].length; j++) // arr[0].length for column
        {
           System.out.print(arr[i][j] + " ");
        }
          System.out.println();
      }
   }
}
```

Output:

```
1 2
3 4
```

Passing Arrays to Methods

Like variables, we can also pass arrays to methods. For example, below program pass

array to method *sum* for calculating sum of array's values.

```
// Java program to demonstrate
// passing of array to method

class Test
{
   // Driver method
   public static void main(String args[])
   {
      int arr[] = {3, 1, 2, 5, 4};

      // passing array to method m1
      sum(arr);

   }

   public static void sum(int[] arr)
   {
      // getting sum of array values
      int sum = 0;

      for (int i = 0; i < arr.length; i++)
         sum+=arr[i];

      System.out.println("sum of array values : " + sum);
   }
}
```

Output :

```
sum of array values : 15
```

Returning Arrays from Methods

As usual, a method can also return an array. For example, below program returns an array from method *m1*.

```
// Java program to demonstrate
// return of array from method


class Test
{
   // Driver method
   public static void main(String args[])
   {
      int arr[] = m1();


      for (int i = 0; i < arr.length; i++)
         System.out.print(arr[i]+" ");


   }


   public static int[] m1()
   {
      // returning  array
      return new int[]{1,2,3};
   }
}
```

Output:

```
1 2 3
```

Class Objects for Arrays

Every array has an associated Class object, shared with all other arrays with the same component type.

```
// Java program to demonstrate
// Class Objects for Arrays


class Test
{
   public static void main(String args[])
   {
      int intArray[] = new int[3];
      byte byteArray[] = new byte[3];
      short shortsArray[] = new short[3];


      // array of Strings
      String[] strArray = new String[3];


      System.out.println(intArray.getClass());
      System.out.println(intArray.getClass().getSuperclass());
      System.out.println(byteArray.getClass());
      System.out.println(shortsArray.getClass());
      System.out.println(strArray.getClass());
   }
}
```

Output:

```
class [I
class java.lang.Object
class [B
class [S
class [Ljava.lang.String;
```

Explanation :

1. The string “[I” is the run-time type signature for the class object “array with component type *int*“.

2. The only direct superclass of any array type is java.lang.Object.
3. The string “[B” is the run-time type signature for the class object “array with component type *byte*“.
4. The string “[S” is the run-time type signature for the class object “array with component type *short*“.
5. The string “[L” is the run-time type signature for the class object “array with component type of a Class”. The Class name is then followed.

Array Members

Now as you know that arrays are object of a class and direct superclass of arrays is class Object. The members of an array type are all of the following:

- The public final field *length*, which contains the number of components of the array. *length* may be positive or zero.
- All the members inherited from class Object; the only method of Object that is not inherited is its clone method.
- The public method *clone()*, which overrides clone method in class Object and throws no checked exceptions.

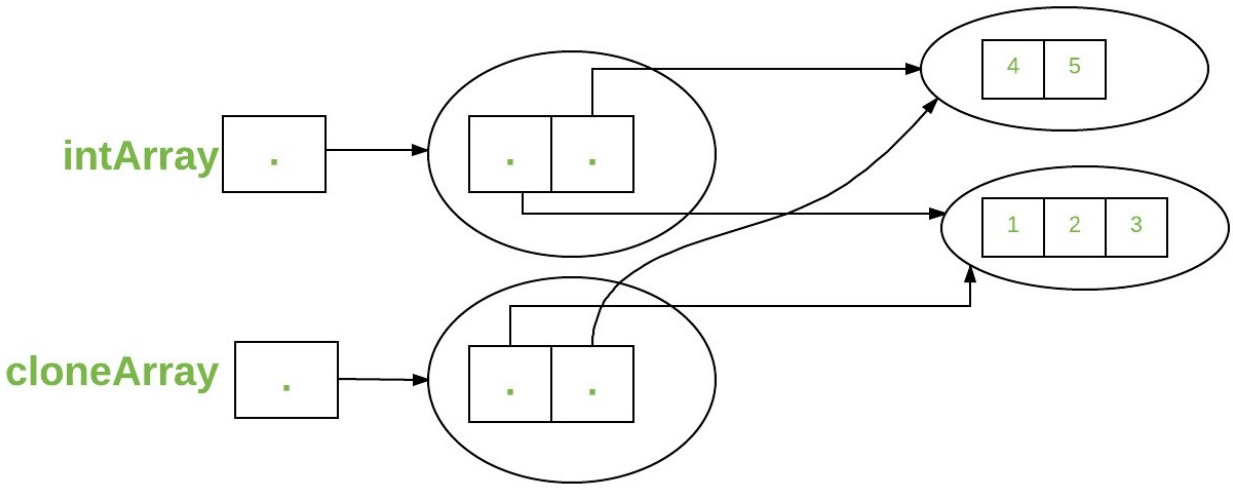
Cloning of arrays

- When you clone a single dimensional array, such as Object[], a “deep copy” is performed with the new array containing copies of the original array’s elements as opposed to references.

```
// Java program to demonstrate
// cloning of one-dimensional arrays
 class Test
{
   public static void main(String args[])
   {
      int intArray[] = {1,2,3};
      int cloneArray[] = intArray.clone();
      // will print false as deep copy is created
      // for one-dimensional array
      System.out.println(intArray == cloneArray);
      for (int i = 0; i < cloneArray.length; i++) {
         System.out.print(cloneArray[i]+" ");
      }
   }}
```

Output:

```
false
1 2 3
```



Shallow Copy is created for multi-dimensional array by clone() method

- A clone of a multidimensional array (like Object[][]) is a “shallow copy” however, which is to say that it creates only a single new array with each element array a reference to an original element array but subarrays are shared.

```
// Java program to demonstrate
// cloning of multi-dimensional arrays


class Test
{
   public static void main(String args[])
   {
      int intArray[][] = {{1,2,3},{4,5}};


      int cloneArray[][] = intArray.clone();


      // will print false
      System.out.println(intArray == cloneArray);


      // will print true as shallow copy is created
      // i.e. sub-arrays are shared
      System.out.println(intArray[0] == cloneArray[0]);
      System.out.println(intArray[1] == cloneArray[1]);


   }
}
```

Output:

```
false
true
true
```

Advantages of using Arrays

- In single statement we can define as many variables as we want.
- Any variable can be accessed using its index number.
- Using for loop we can access each element of any array.
- Program become short and therefore, easy to compile and debug.

Copying Arrays

We can use **arraycopy( )** method of **System** class to efficiently copy data from one array into another.

The **arraycopy()** method requires five arguments as shown below:

public static void arracopy(Object source, int srcIndrx, Object dest, int destIndex, int length)

The two Object argument indicate the array to copy from and the array to copy to. The three integer arguments indicate the starting location in each, the source and the destination array, and the number of elements to copy.

Example:

```
int arr1[ ]= {23,45,67,78,12,89,90,34,32,54,76,87,98,21};
int arr2[ ]=new int [10];
System.arraycopy(arr1, 4, arr2, 0, 5);
```

value of arr2 is {12,89,90,34,32}

arr1 is source and arr2 is destination arr2 copy the 5 elements from 0 index of arr1 from 4 index.

Write a program to enter 10 numbers and print sum all number.

```
import java.util.*;
class Program
	{
		public static void main(String [] args)
		{
			Scanner sn=new Scanner (System.in);
			int arr[ ] = new int[10];
			int i,sum=0;
			for(i=0;i<=9;i++)
				{
					System.out.print(“Enter number= ”);
					arr[i]=sn.nextInt();
				}
			for(i=0;i<=9;i++)
				{
					sum=sum+arr[i];
				}
		System.out.println(“Sum of All Numbers= ”+ sum);
		}
	}
```