Arrays

An *array* is a collection of variables of the same type, referred to by a common name. In Java, arrays can have one or more dimensions, although the one-dimensional array is the most common. Arrays are used for a variety of purposes because they offer a convenient means of grouping together related variables.

For example, you might use an array to hold a record of the daily high temperature for a month, a list of stock price averages, or a list of your collection of programming books.

The principal advantage of an array is that it organizes data in such a way that it can be easily manipulated. For example, if you have an array containing the incomes for a selected group of households, it is easy to compute the average income by cycling through the array. In Java Arrays are implemented as objects.

One-Dimensional Arrays

A one-dimensional array is a list of related variables. For example, you might use a one-dimensional array to store the account numbers of the active users on a network. Another array might be used to store the current batting averages for a cricket team.

Declaration of one Dimensional Arrays:

type array-name[ ] = new *type*[*size*];

1. Here, *type* declares the element type of the array.
2. The element type determines the data type of each element contained in the array.
3. The number of elements that the array will hold is determined by *size.*
4. Since arrays are implemented as objects, the creation of an array is a two-step process. First, you declare an array reference variable. Second, you allocate memory for the array, assigning a reference to that memory to the array variable.
5. Thus, arrays in Java are dynamically allocated using the **new** operator.

Here is an example. The following creates an **int** array of 10 elements and links it to an array reference variable named **sample**:

```
int sample[] = new int[10];
```

This declaration works just like an object declaration. The **sample** variable holds a reference to the memory allocated by **new**. This memory is large enough to hold 10 elements of type **int**. As with objects, it is possible to break the preceding declaration in two. For example:

```
int sample[]; sample = new int[10];
```

In this case, when **sample** is first created, it refers to no physical object. It is only after the second statement executes that **sample** is linked with an array.

An individual element within an array is accessed by use of an index. An *index* describes the position of an element within an array. In Java, all arrays have zero as the index of their first element. Because **sample** has 10 elements, it has index values of 0 through 9. To index an array, specify the number of the element you want, surrounded by square brackets. Thus, the first element in **sample** is **sample[0]**, and the last element is **sample[9]**.

Example:-

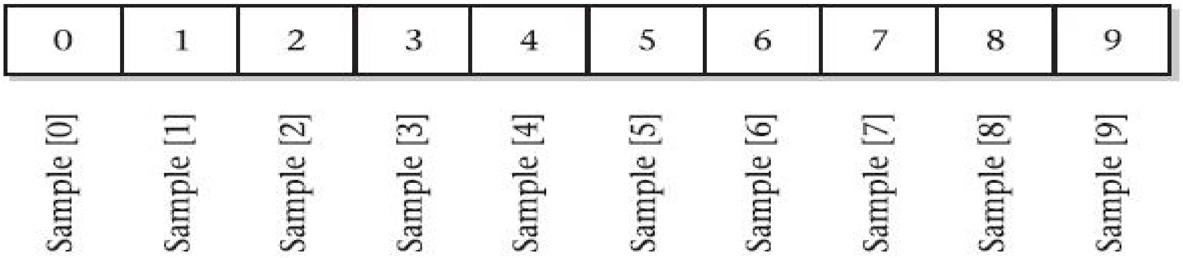
```
public class Arrays1 {
   public static void main(String args[]) {
      int sample[] = new int[10];
      int i;
      for(i = 0; i < 10; i = i+1)
         sample[i] = i;
      for(i = 0; i < 10; i = i+1)
         System.out.println("This is sample[" + i + "]: " + sample[i]);
   }
}
```

Output:

The output from the program is shown here:

```
This is sample[0]: 0
This is sample[1]: 1
This is sample[2]: 2
This is sample[3]: 3
This is sample[4]: 4
This is sample[5]: 5
This is sample[6]: 6
This is sample[7]: 7
This is sample[8]: 8
This is sample[9]: 9
```

Conceptually, the **sample** array looks like this:



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]);
```

Example: **Java Program to find Min and Max element in user given array.**

```
import java.util.Scanner;
class MinMax {
   public static void main(String args[]) {
      Scanner scan=new Scanner(System.in);
      System.out.println(“Enter Size of the Array: “);
      int len=scan.nextInt();
      int nums[] = new int[len];
      int min,max;
      for(int i=0;i<len;i++)
      {
         System.out.print("Enter"+i+"th element: ");
         nums[i]=scan.nextInt();
       }
      min=max=nums[0];
      for(int i=0;i<len;i++){
         if(min>nums[i])
            min=nums[i];
         else
            max=nums[i];
      }
      System.out.println("Min element is: "+min+"\nMax element is: "+max);
   }
}
```

Multidimensional arrays

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

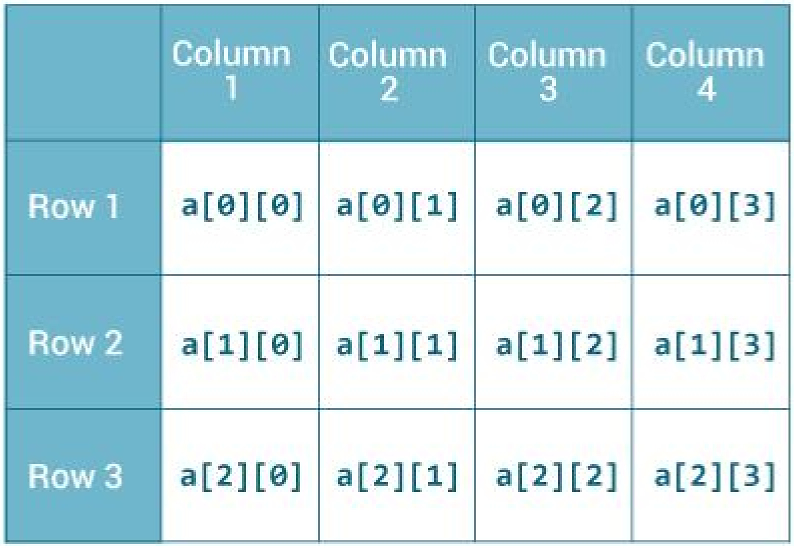
Examples:

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

For example,

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

Here, a is a two-dimensional (2d) array. The array can hold maximum of 12 elements of type int.



	Column 1	Column 2	Column 3	Column 4
Row 1	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 2	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 3	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Similarly, you can declare a three-dimensional (3d) array. For example,

```
String[][][] personalInfo = new String[3][4][2];
```

Here, *personalInfo* is a 3d array that can hold maximum of 24 (3*4*2) elements of type String.

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

    int[][] a = {
        {1, 2, 3},
        {4, 5, 6, 9},
        {7},
    };

    System.out.println("Length of row 1: " + a[0].length);
    System.out.println("Length of row 2: " + a[1].length);
    System.out.println("Length of row 3: " + a[2].length);
  }
}
```

When you run the program, the output will be:

```
Length of row 1: 3
Length of row 2: 4
Length of row 3: 1
```

Since each component of a multidimensional array is also an array (a[0], a[1] and a[2] are also arrays), you can use length attribute to find the length of each rows.

Example: Print all elements of 2d array Using Loop

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

    int[][] a = {
        {1, -2, 3},
        {-4, -5, 6, 9},
        {7},
    };

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

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
```

Example: Multiply two Matrices

```
import java.util.Scanner;
public class JavaProgram
{
  public static void main(String args[])
  {
    int m, n, p, q, sum = 0, c, d, k;
    Scanner in = new Scanner(System.in);
    System.out.print("Enter Number of Rows and Columns of First Matrix : ");
    m = in.nextInt();
    n = in.nextInt();
    int first[][] = new int[m][n];
    System.out.print("Enter First Matrix Elements : ");
    for(c=0 ; c<m; c++)
    {
      for(d=0; d<n; d++)
      {
        first[c][d] = in.nextInt();
      }
    }

    System.out.print("Enter Number of Rows and Columns of Second Matrix : ");
    p = in.nextInt();
    q = in.nextInt();

    if ( n != p )
    {
      System.out.print("Matrix of the entered order can't be Multiplied..!!");
    }
    else
    {
      int second[][] = new int[p][q];
      int multiply[][] = new int[m][q];

      System.out.print("Enter Second Matrix Elements :\n");

      for(c=0; c<p; c++)
      {
        for(d=0; d<q; d++)
        {
          second[c][d] = in.nextInt();
        }
      }

      System.out.print("Multiplying both Matrix...\n");

      for(c=0; c<m; c++)
      {
        for(d=0; d<q; d++)
        {
          for(k=0; k<p; k++)
          {
            sum = sum + first[c][k]*second[k][d];
          }

          multiply[c][d] = sum;
          sum = 0;
        }
      }

      System.out.print("Multiplication Successfully performed..!!\n");
      System.out.print("Now the Matrix Multiplication Result is :\n");

      for(c=0; c<m; c++)
      {
        for(d=0; d<q; d++)
        {
          System.out.print(multiply[c][d] + "\t");
        }
        System.out.print("\n");
      }
    }

  }
}
```