

Array

1. Declaring an Array

```
int c[] = new int[ 12 ];
```

Above task in two steps

```
int c[];           // declare the array variable  
c = new int[ 12 ]; // create the array; assign to array variable
```

The diagram illustrates an array structure. On the left, the text 'Name of array (c)' has an arrow pointing to the first row of the table. Below it, the text 'Index (or subscript) of the element in array c' has an arrow pointing to the index column of the table. The table itself has two columns: the first column contains indices from 0 to 11, and the second column contains corresponding integer values.

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

A program can create several arrays in a single declaration. The following String array declaration reserves 100 elements for b and 27 elements for x:

```
String b[] = new String[ 100 ], x[] = new String[ 27 ];
```

In this case, the class name String applies to each variable in the declaration. For readability, we prefer to declare only one variable per declaration, as in:

```
String b[] = new String[ 100 ]; // create array b  
String x[] = new String[ 27 ]; // create array x
```

When an array is declared, the type of the array and the square brackets can be combined at the beginning of the declaration to indicate that all the identifiers in the declaration are array variables. For example, the declaration

```
double[] array1, array2;
```

indicates that `array1` and `array2` are each “array of `double`” variables. The preceding declaration is equivalent to:

```
double array1[];  
double array2[];
```

or

```
double[] array1;  
double[] array2;
```

The preceding pairs of declarations are equivalent—when only one variable is declared in each declaration, the square brackets can be placed either after the type or after the array variable name.

Example:

```
1 // Fig. 7.2: InitArray.java  
2 // Creating an array.  
3  
4 public class InitArray  
5 {  
6     public static void main( String args[] )  
7     {  
8         int array[]; // declare array named array  
9  
10        array = new int[ 10 ]; // create the space for array  
11  
12        System.out.printf( "%s%8s\n", "Index", "Value" ); // column headings  
13  
14        // output each array element's value  
15        for ( int counter = 0; counter < array.length; counter++ )  
16            System.out.printf( "%5d%8d\n", counter, array[ counter ] );  
17    } // end main  
18 } // end class InitArray
```

Index	Value
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

Example

```
1 // Fig. 7.3: InitArray.java
2 // Initializing the elements of an array with an array initializer.
3
4 public class InitArray
5 {
6     public static void main( String args[] )
7     {
8         // initializer list specifies the value for each element
9         int array[] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
10
11         System.out.printf( "%s%s\n", "Index", "Value" ); // column headings
12
13         // output each array element's value
14         for ( int counter = 0; counter < array.length; counter++ )
15             System.out.printf( "%5d%8d\n", counter, array[ counter ] );
16     } // end main
17 } // end class InitArray
```

Index	Value
0	32
1	27
2	64
3	18
4	95
5	14
6	90
7	70
8	60
9	37

Fig. 7.3 | Initializing the elements of an array with an array initializer.

Example

```
1 // Fig. 7.4: InitArray.java
2 // Calculating values to be placed into elements of an array.
3
4 public class InitArray
5 {
6     public static void main( String args[] )
7     {
8         final int ARRAY_LENGTH = 10; // declare constant
9         int array[] = new int[ ARRAY_LENGTH ]; // create array
10
11         // calculate value for each array element
12         for ( int counter = 0; counter < array.length; counter++ )
13             array[ counter ] = 2 + 2 * counter;
14
15         System.out.printf( "%s%s\n", "Index", "Value" ); // column headings
16
17         // output each array element's value
18         for ( int counter = 0; counter < array.length; counter++ )
19             System.out.printf( "%5d%8d\n", counter, array[ counter ] );
20     } // end main
21 } // end class InitArray
```

Index	Value
0	2
1	4
2	6
3	8
4	10
5	12
6	14
7	16
8	18
9	20

Fig. 7.4 | Calculating the values to be placed into the elements of an array.

2. Using an Array initializer

```
int n[] = {10, 20, 30, 40, 50}
```

or

```
int[] anArray; // declares an array of integers
String name[] = {"John", "Peter", "Maria"};
String[] name = {"John", "Ali", "Maria"};
String[] name = new String[] {"Oscar", "Charlie", "Ryan", "Adam"}
Similarly, you can declare arrays of other types:
```

```
float[] anArrayOfFloats;
double[] anArrayOfDoubles;
boolean[] anArrayOfBooleans;
char[] anArrayOfChars;
```

```
String[] anArrayOfStrings;
```

You can also place the square brackets after the array's name:

```
float anArrayOfFloats[]; // this form is discouraged
```

However, convention discourages this form; the brackets identify the array type and should appear with the type designation.

3. Sorting an Array

```
import java.util.Arrays;

public class Sort {
    public void sortStringArray()
    {
        String[] arrayToSort = new String[] {"Oscar", "Charlie",
        "Ryan", "Adam", "David"};

        //To sort an array
        Arrays.sort(arrayToSort);

        //Display sorted array
        for (int i = 0; i < arrayToSort.length; i++)
            System.out.println(arrayToSort[i]);
    }
}
```

Main entry point class

```
public class SortTest{
    public static void main(String args[]) {
        Sort obj = new Sort();
        obj.sortStringArray();
    }
}
```

4. Passing Arrays to Methods

```
1 // Fig. 7.13: PassArray.java
2 // Passing arrays and individual array elements to methods.
3
4 public class PassArray
5 {
6     // main creates array and calls modifyArray and modifyElement
7     public static void main( String args[] )
8     {
9         int array[] = { 1, 2, 3, 4, 5 };
10
11         System.out.println(
12             "Effects of passing reference to entire array:\n" +
13             "The values of the original array are:" );
14
15         // output original array elements
16         for ( int value : array )
17             System.out.printf( "    %d", value );
18
19         modifyArray( array ); // pass array reference
20         System.out.println( "\n\nThe values of the modified array are:" );
21
22         // output modified array elements
23         for ( int value : array )
24             System.out.printf( "    %d", value );
25
26         System.out.printf(
27             "\n\nEffects of passing array element value:\n" +
28             "array[3] before modifyElement: %d\n", array[ 3 ] );
29
30         modifyElement( array[ 3 ] ); // attempt to modify array[ 3 ]
31         System.out.printf(
32             "array[3] after modifyElement: %d\n", array[ 3 ] );
33     } // end main
34
35     // multiply each element of an array by 2
36     public static void modifyArray( int array2[] )
37     {
38         for ( int counter = 0; counter < array2.length; counter++ )
39             array2[ counter ] *= 2;
40     } // end method modifyArray
41
42     // multiply argument by 2
43     public static void modifyElement( int element )
44     {
45         element *= 2;
46         System.out.printf(
47             "Value of element in modifyElement: %d\n", element );
48     } // end method modifyElement
49 } // end class PassArray
```

```

Effects of passing reference to entire array:
The values of the original array are:
  1  2  3  4  5

The values of the modified array are:
  2  4  6  8  10

Effects of passing array element value:
array[3] before modifyElement: 8
Value of element in modifyElement: 16
array[3] after modifyElement: 8

```

5. Multi-dimensional Arrays

Multidimensional arrays with two dimensions are often used to represent **tables of values** consisting of information arranged in **rows** and **columns**. To identify a particular table element, we must specify two indices. By convention, the first identifies the element's row and the second its column. Arrays that require two indices to identify a particular element are called **two-dimensional arrays**. (Multidimensional arrays can have more than two dimensions.) Java does not support multidimensional arrays directly, but it does allow the programmer to specify one-dimensional arrays whose elements are also one-dimensional

thus, achieving the same effect.

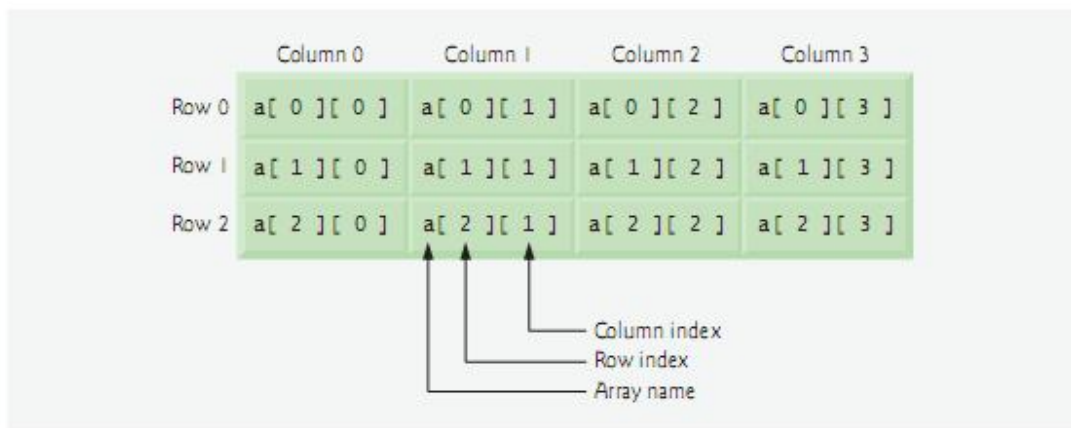


Fig. 7.16 | Two-dimensional array with three rows and four columns.

A two dimensional array b with two rows and two columns could be declared and initialized with nest array initializer as follows.

```
Int b[ ][ ]= { {1,2}, [3,4] };
```

Two dimensional array with rows of different lengths could be declared and initialized as follows.

```
Int b[ ][ ]= { {1,2}, {3,4, 5} };
```



```

1 // Fig. 7.17: InitArray.java
2 // Initializing two-dimensional arrays.
3
4 public class InitArray
5 {
6     // create and output two-dimensional arrays
7     public static void main( String args[] )
8     {
9         int array1[][] = { { 1, 2, 3 }, { 4, 5, 6 } };
10        int array2[][] = { { 1, 2 }, { 3 }, { 4, 5, 6 } };
11
12        System.out.println( "Values in array1 by row are" );
13        outputArray( array1 ); // displays array1 by row
14
15        System.out.println( "\nValues in array2 by row are" );
16        outputArray( array2 ); // displays array2 by row
17    } // end main
18
19    // output rows and columns of a two-dimensional array
20    public static void outputArray( int array[][] )
21    {
22        // loop through array's rows
23        for ( int row = 0; row < array.length; row++ )
24        {
25            // loop through columns of current row
26            for ( int column = 0; column < array[ row ].length; column++ )
27                System.out.printf( "%d ", array[ row ][ column ] );
28
29            System.out.println(); // start new line of output
30        } // end outer for
31    } // end method outputArray
32 } // end class InitArray

```

Values in array1 by row are

```

1 2 3
4 5 6

```

Values in array2 by row are

```

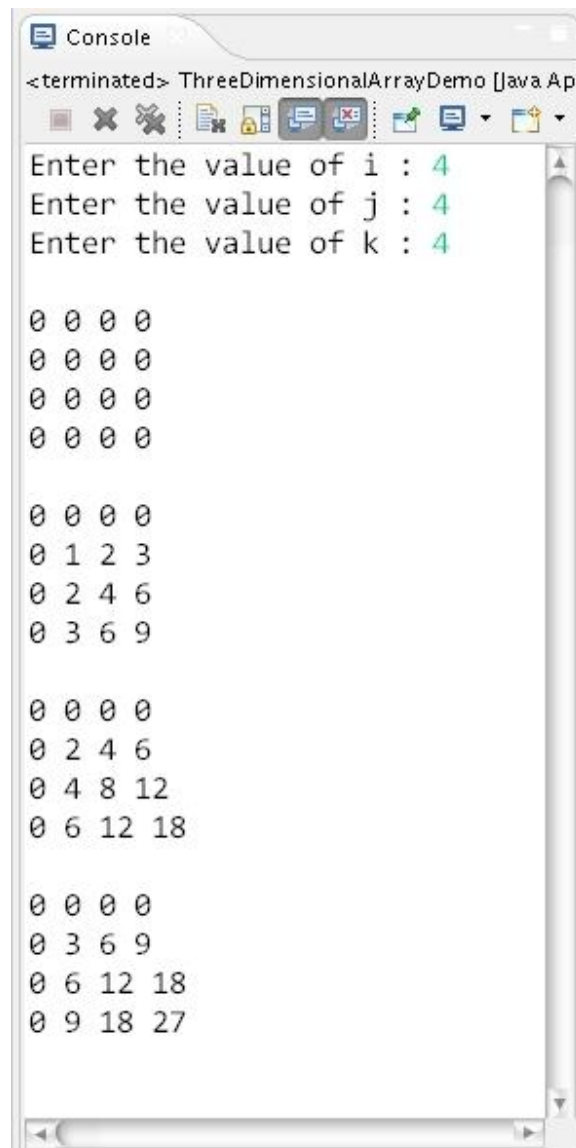
1 2
3
4 5 6

```


Program to demonstrate three dimensional Array in Java

```
2
1
2
3 import java.util.Scanner;
4
5 public class ThreeDimensionalArrayDemo {
6     public static void main(String args[]) {
7
8         Scanner input = new Scanner(System.in);
9
10        int threeDArray[][][] = new int[10][10][10];
11        int i = 0, j = 0, k = 0;
12
13        System.out.print("Enter the value of i : ");
14        i = input.nextInt();
15
16        System.out.print("Enter the value of j : ");
17        j = input.nextInt();
18
19        System.out.print("Enter the value of k : ");
20        k = input.nextInt();
21
22        System.out.println("");
23
24        for(int m=0; m < i; m++)
25            for(int n=0; n < j; n++)
26                for(int p=0; p < k; p++)
27                    threeDArray[m][n][p] = m * n * p;
28
29        for(int m=0; m < i; m++) {
30            for(int n=0; n < j; n++) {
31                for(int p=0; p < k; p++)
32                    System.out.print(threeDArray[m][n][p] + " ");
33                System.out.println();
34            }
35        }
36        System.out.println();
37    }
38 }
```

Output of the program :



The screenshot shows a Java IDE console window titled "Console" with a tab for "ThreeDimensionalArrayDemo [Java Ap". The console displays the following text:

```
<terminated> ThreeDimensionalArrayDemo [Java Ap
Enter the value of i : 4
Enter the value of j : 4
Enter the value of k : 4

0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0

0 0 0 0
0 1 2 3
0 2 4 6
0 3 6 9

0 0 0 0
0 2 4 6
0 4 8 12
0 6 12 18

0 0 0 0
0 3 6 9
0 6 12 18
0 9 18 27
```

The output consists of three input prompts followed by three sets of 4x4 grids of numbers. Each grid represents a slice of a 3D array. The first grid is all zeros. The second grid shows a sequence of numbers from 0 to 9. The third grid shows a sequence of numbers from 0 to 27.

Collection

- *A collection* — sometimes called a container — is simply an object that groups multiple elements into a single unit.
- Collections are used to store, retrieve, manipulate, and communicate aggregate data.
- Typically, they represent data items that form a natural group, such as a poker hand (a collection of cards), a mail folder (a collection of letters), or a telephone directory (a mapping of names to phone numbers).

Benefits of the Java Collections Framework

- Reduces programming effort
- Increases program speed and quality
- Fosters software reuse
- Many more

```
import java.util.*;
```

```
public class CollectionDemo {
```

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

```
        List a1 = new ArrayList();
```

```
        a1.add("Zara");
```

```
        a1.add("Mahnaz");
```

```
        a1.add("Ayan");
```

```
        System.out.println(" ArrayList Elements");
```

```
        System.out.print("\t" + a1);
```

```
    }
```

```
}
```

Output

ArrayList Elements

[Zara, Mahnaz, Ayan]

```
import java.util.List;
class Student{

    static int counterSDP, counterDDP;
    double calculateFee(){
        System.out.println("fee is not defined");
        return 0.0;
    }

    public static void calculateFee(List<Student> student){
        for(Student i: student){
            if(i instanceof SingleDegreeStudent)
            {
                System.out.println("Fee of Single Degree Student is : "+i.calculateFee());
                counterSDP++;
            }
            else
            {
                System.out.println("Fee of DDP Student is : "+i.calculateFee());
                counterDDP++;
            }
        }
    }
}
```

```
static void printNumberOfStudents()
{
    System.out.println("Number of SDP Student="+counterSDP);
    System.out.println("Number of DDP Student="+counterDDP);
}
}
```



```
class SingleDegreeStudent extends Student{  
    private int noOfcreditHours;  
  
    public SingleDegreeStudent (int noOfcreditHours){  
        this.noOfcreditHours=noOfcreditHours;  
    }  
  
    double calculateFee()  
    {  
        return noOfcreditHours * 6000 + 45000;  
    }  
}
```

```
class DDPStudent extends Student{  
    private int noOfcreditHours;  
  
    public DDPStudent (int noOfcreditHours){  
        this.noOfcreditHours=noOfcreditHours;  
    }  
  
    double calculateFee()  
    {  
        return noOfcreditHours * 10000 + 90000 ;  
    }  
}
```

```
import java.util.ArrayList;
import java.util.List;

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

        List<Student> stu = new ArrayList<Student>();

        stu.add(new SingleDegreeStudent(135));
        stu.add(new SingleDegreeStudent(130));

        stu.add(new DDPStudent(135));
        stu.add(new DDPStudent(130));
        stu.add(new DDPStudent(140));

        double sumOfFees = 0.0;
        for(Student i: stu){
            sumOfFees += i.calculateFee();
        }
        System.out.println( "Sum of Fees = " + sumOfFees);

        Student.calculateFee(stu);
        Student.printNumberOfStudents();

    }
}
```

OUTPUT

Sum of Fees = 6000000.0

Fee of Single Degree Student is : 855000.0

Fee of Single Degree Student is : 825000.0

Fee of DDP Student is : 1440000.0

Fee of DDP Student is : 1390000.0

Fee of DDP Student is : 1490000.0

Number of SDP Student=2

Number of DDP Student=3

THANK YOU