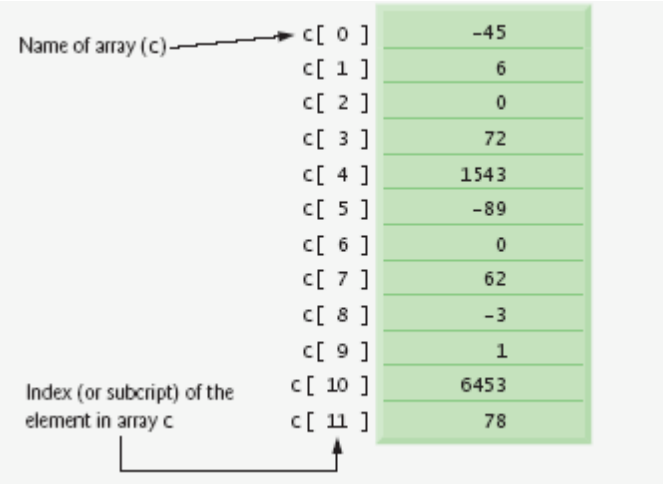
Array

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
```



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%8s\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%8s\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.

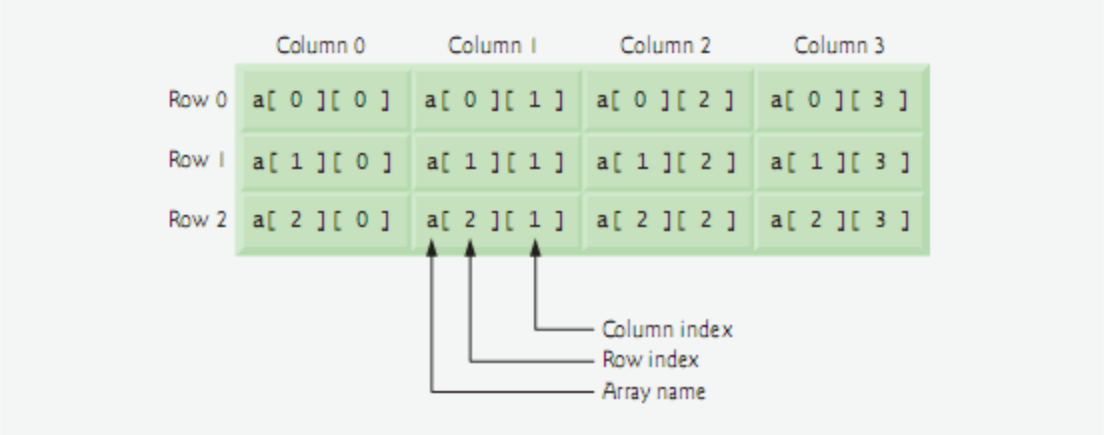


Fig. 7.16 | Two-dimensional array with three rows and four columns.

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} };

```java
// Fig. 7.17: InitArray.java
// Initializing two-dimensional arrays.

public class InitArray
{
   // create and output two-dimensional arrays
   public static void main( String args[] )
   {
      int array1[][] = { { 1, 2, 3 }, { 4, 5, 6 } };
      int array2[][] = { { 1, 2 }, { 3 }, { 4, 5, 6 } };

      System.out.println( "Values in array1 by row are" );
      outputArray( array1 ); // displays array1 by row

      System.out.println( "\nValues in array2 by row are" );
      outputArray( array2 ); // displays array2 by row
   } // end main

   // output rows and columns of a two-dimensional array
   public static void outputArray( int array[][] )
   {
      // loop through array's rows
      for ( int row = 0; row < array.length; row++ )
      {
         // loop through columns of current row
         for ( int column = 0; column < array[ row ].length; column++ )
            System.out.printf( "%d  ", array[ row ][ column ] );

         System.out.println(); // start new line of output
      } // end outer for
   } // end method outputArray
} // 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
```