

Multidimensional Arrays

Arrays that have exactly two indices can be displayed on paper as a two-dimensional table and are called two-dimensional arrays. The Java notation for an element of a two-dimensional array is

`Array_Name[Row_Index][Column_Index]`

For example, if the array is named `table` and it has two indices, `table[3][2]` is the entry whose row index is 3 and column index is 2. Because indices begin at 0, this entry is in the fourth row and third column of `table`.

Arrays having multiple indices are handled in much the same way as one-dimensional arrays. The following statement declares the name `table` and creates a two-dimensional array :

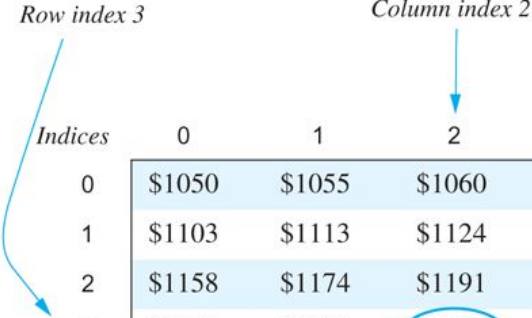
```
int[ ][ ] table = new int[10][6];
```

This is equivalent to the following two statements:

```
int[ ][ ] table;
```

```
table = new int[10][6];
```

Row and Column Indices for an Array Named `table`



Indices	0	1	2	3	4	5
0	\$1050	\$1055	\$1060	\$1065	\$1070	\$1075
1	\$1103	\$1113	\$1124	\$1134	\$1145	\$1156
2	\$1158	\$1174	\$1191	\$1208	\$1225	\$1242
3	\$1216	\$1239	\$1262	\$1286	\$1311	\$1335
4	\$1276	\$1307	\$1338	\$1370	\$1403	\$1436
5	\$1340	\$1379	\$1419	\$1459	\$1501	\$1543
6	\$1407	\$1455	\$1504	\$1554	\$1606	\$1659
7	\$1477	\$1535	\$1594	\$1655	\$1718	\$1783
8	\$1551	\$1619	\$1689	\$1763	\$1838	\$1917
9	\$1629	\$1708	\$1791	\$1877	\$1967	\$2061

`table[3][2]` has a value of 1262

Arrays having more than one index are generally called multidimensional arrays. More specifically, an array that has n indices is said to be an n -dimensional array. Thus, an ordinary one-index array is a one-dimensional array. Although arrays having more than two dimensions are rare, they can be useful for some applications.

Example 7.12

```
/**
 * Displays a two-dimensional table showing how
 * interest rates affect bank balances.
 */
public class InterestTable
{
    public static void main(String[] args)
    {
        int[][] table = new int[10][6];
        for (int row = 0; row < 10; row++)
            for (int column = 0; column < 6; column++)
                table[row][column] =
                    getBalance(1000.00, row + 1, (5 + 0.5 *
                                                                column));

        System.out.println("Balances for Various Interest Rates " +
                           "Compounded Annually");
        System.out.println("(Rounded to Whole Dollar Amounts)");
        System.out.println();
        System.out.println("Years 5.00% 5.50% 6.00% 6.50% " +
                           "7.00% 7.50%");
        for (int row = 0; row < 10; row++)
        {
            System.out.print((row + 1) + " ");
            for (int column = 0; column < 6; column++)
                System.out.print("$" + table[row][column] + " ");
            System.out.println();
        }
    }
}

/**
 * Returns the balance in an account after a given number of years
 * and interest rate with an initial balance of startBalance.
 * Interest is compounded annually. The balance is rounded
 * to a whole number.
 */
public static int getBalance(double startBalance, int years,
                             double rate)
{
    double runningBalance = startBalance;
    for (int count = 1; count <= years; count++)
        runningBalance = runningBalance * (1 + rate / 100);
    return (int)(Math.round(runningBalance));
}
}
```

A real application would do something more with the array `table`. This is just a demonstration program.

Sample Screen Output

Balances for Various Interest Rates Compounded Annually (Rounded to Whole Dollar Amounts)

Years	5.00%	5.50%	6.00%	6.50%	7.00%	7.50%
1	\$1050	\$1055	\$1060	\$1065	\$1070	\$1075
2	\$1103	\$1113	\$1124	\$1134	\$1145	\$1156
3	\$1158	\$1174	\$1191	\$1208	\$1225	\$1242
4	\$1216	\$1239	\$1262	\$1286	\$1311	\$1335
5	\$1276	\$1307	\$1338	\$1370	\$1403	\$1436
6	\$1340	\$1379	\$1419	\$1459	\$1501	\$1543
7	\$1407	\$1455	\$1504	\$1554	\$1606	\$1659
8	\$1477	\$1535	\$1594	\$1655	\$1718	\$1783
9	\$1551	\$1619	\$1689	\$1763	\$1838	\$1917
10	\$1629	\$1708	\$1791	\$1877	\$1967	\$2061

The last line is out of alignment because 10 has two digits. This is easy to fix, but that would clutter the discussion of arrays with extraneous concerns.

Note that this syntax is almost identical to the syntax we used for the one-dimensional case. The only difference is that we added a second pair of square brackets in two places, and we gave a number specifying the size of the second dimension, that is, the number of columns. You can have arrays with any number of indices. To get more indices, you just use more square brackets in the declaration.

Indexed variables for multidimensional arrays are just like indexed variables for one-dimensional arrays, except that they have multiple indices, each enclosed in a pair of square brackets.

```

/**
 * Displays a two-dimensional table showing how interest
 * rates affect bank balances.
 */
public class InterestTable2
{
    public static final int ROWS = 10;
    public static final int COLUMNS = 6;

    public static void main(String[] args)
    {
        int[][] table = new int[ROWS][COLUMNS];
        for (int row = 0; row < ROWS; row++)
            for (int column = 0; column < COLUMNS; column++)
                table[row][column] =
                    getBalance(1000.00, row + 1, (5 + 0.5 * column));

        System.out.println("Balances for Various Interest Rates " +
                           "Compounded Annually");
        System.out.println("(Rounded to Whole Dollar Amounts)");
        System.out.println();
        System.out.println("Years 5.00% 5.50% 6.00% 6.50% 7.00% 7.50%");

        showTable(table);
    }
}
/**
 * Precondition: The array anArray has ROWS rows and COLUMNS columns.
 * Postcondition: The array contents are displayed with dollar signs.
 */
public static void showTable(int[][] anArray)
{
    for (int row = 0; row < ROWS; row++)
    {
        System.out.print((row + 1) + " ");
        for (int column = 0; column < COLUMNS; column++)
            System.out.print("$" + anArray[row][column] + " ");
        System.out.println();
    }
}

public static int getBalance(double startBalance, int years, double rate)
<The rest of the definition of getBalance is the same as in Listing 7.12.>
}

```

A better definition of
showTable is possible,
as you will see.

The output is the same
as in Listing 7.12.

The Java compiler represents a multidimensional array as several one-dimensional arrays. For example, consider the two-dimensional array

```
int[ ][ ] table = new int[10][6];
```

The array table is in fact a one-dimensional array of length 10, and its base type is the type int[]. In other words, multidimensional arrays are arrays of arrays.

When using length with a multidimensional array, you need to think in terms of arrays of arrays.

Using length for multidimensional arrays

```
for (int row = 0; row <table.length; row++)  
    for (int column = 0;column< table[row].length; column++)  
        table[row][column]= getBalance(1000.00,row + 1,(5 + 0.5 * column));
```

Since the array table is actually a one-dimensional array of length 10, the first for loop can use table.length instead of ROWS, which is 10. Additionally, each of the 10 indexed variables table[0]through table[9] is a one-dimensional array whose length is 6 (COLUMNS). Thus, since table[row] is a one-dimensional array whose length is 6, the second for loop can use table[row].length instead of COLUMNS.

For a two-dimensional array b, the value of b.length is the number of rows, that is, the integer in the first pair of brackets in the array's declaration. The value of b[i].length is the number of columns, that is, the integer in the second pair of brackets in the array's declaration.