Arrays

Suppose we wanted to store fifty values in fifty variables. We could just create different variables using different names.

|  |  |
| --- | --- |
| number1 |  |
| number2 |  |
| . |  |
| . |  |
| number50 |  |

double number1;

double number2;

:

double number50;

But even with copy-and-paste, this is terribly inconvenient. We would need fifty commands to initialize each of our variables to zero.

|  |  |
| --- | --- |
| number1 | 0.0 |
| number2 | 0.0 |
| . |  |
| . |  |
| number3 | 0.0 |

number1 = 0.0;

number2 = 0.0;

:

number50 = 0.0;

This situation begs for an *array*, which is a collection of variables under one name, where the variables are accessed by *index* numbers. As an analogy, arrays are like lockers and the locker numbers are like index numbers. You access each locker, or *cell*, by knowing its number. In our example there are fifty cells. Each cell is numbered, starting from zero. The integer numbering each cell, in the square brackets, is that cell’s index (or its location, or position, or address).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| myArray |  |  |  |  |  |  | ... |  |
|  |  |  | [0] | [1] | [2] | [3] | ... | [49] |

**double**[] myArray = **new** **double**[50]; //declares an empty array

**for**(**int** k = 0; k < 50; k++) //the **for**-loop visits each cell

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| myArray |  |  | 0.0 | 0.0 | 0.0 | 0.0 | **. . .** | 0.0 |  | the data |
|  |  |  | [0] | [1] | [2] | [3] | **. . .** | [49] |  | the index numbers |

myArray[k] = 0.0; //and assigns 0.0 to each cell

In Java, arrays are objects requiring the keyword new. The data in an array is accessed through the array’s name, which is a reference, and the index number in square brackets. Arrays are usually processed by for-loops.

Be careful! Arrays are zero-indexed so the first cell is cell number zero and the fiftieth cell is cell number forty-nine. You cannot change this fact so you must either adjust your thinking or accept frustration.

The name of an array does not have to be *array*, the type of data stored in an array does not have to be *double*, and the size of an array does not have to be *50*. Here are some examples of array declarations that you may see:

**int**[] scores = **new** **int**[18];

**char**[] alphabet = **new** **char**[26];

String[] words = **new** String[NUMITEMS]; **//NUMITEMS must have been initialized**

You access individual items of data by specifying the index of the cell. Show the effects of these commands:

**int**[] theArray = **new int**[6];

theArray[4] = 5;

theArray[1] = 3;

theArray[4] = theArray[1];

theArray[1] = theArray[1] \* 2;

System.out.println("" + theArray[1]);

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| theArray |  |  |  |  |  |  |  |  |
|  |  |  | [0] | [1] | [2] | [3] | [4] | [5] |

Exercises

1) Draw a picture of what this declaration accomplishes:

**int**[] scores = **new** **int**[5];

2) Write the code to fill the array above with 10.

3) Draw a picture of what this declaration accomplishes:

**char**[] answers = **new** **char**[4];

4) Write the code to fill each cell in the array above with an 'A'.

5) Write the code to print each element of the answers array in the console I/O window.

6) Given an array prices containing doubles, write the code to multiply each element by 10.0.