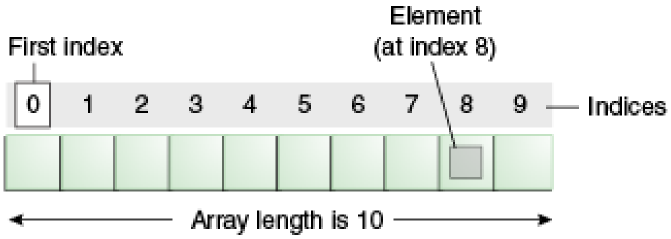
Trail: Learning the Java Language
Lesson: Language Basics
Section: Variables

Arrays

An *array* is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed. You've seen an example of arrays already, in the `main` method of the "Hello World!" application. This section discusses arrays in greater detail.



An array of ten elements

Each item in an array is called an *element*, and each element is accessed by its numerical *index*. As shown in the above illustration, numbering begins with 0. The 9th element, for example, would therefore be accessed at index 8.

The following program, [`ArrayDemo`](), creates an array of integers, puts some values in it, and prints each value to standard output.

```
class ArrayDemo {
    public static void main(String[] args) {
        // declares an array of integers
        int[] anArray;

        // allocates memory for 10 integers
        anArray = new int[10];

        // initialize first element
        anArray[0] = 100;
        // initialize second element
        anArray[1] = 200;
        // etc.
        anArray[2] = 300;
        anArray[3] = 400;
        anArray[4] = 500;
        anArray[5] = 600;
        anArray[6] = 700;
        anArray[7] = 800;
        anArray[8] = 900;
        anArray[9] = 1000;
        System.out.println("Element at index 0: "
                           + anArray[0]);
        System.out.println("Element at index 1: "
                           + anArray[1]);
        System.out.println("Element at index 2: "
                           + anArray[2]);
        System.out.println("Element at index 3: "
                           + anArray[3]);
        System.out.println("Element at index 4: "
                           + anArray[4]);
        System.out.println("Element at index 5: "
                           + anArray[5]);
        System.out.println("Element at index 6: "
                           + anArray[6]);
        System.out.println("Element at index 7: "
                           + anArray[7]);
        System.out.println("Element at index 8: "
                           + anArray[8]);
        System.out.println("Element at index 9: "
                           + anArray[9]);
    }
}
```

The output from this program is:

```
Element at index 0: 100
Element at index 1: 200
Element at index 2: 300
Element at index 3: 400
Element at index 4: 500
Element at index 5: 600
Element at index 6: 700
Element at index 7: 800
Element at index 8: 900
Element at index 9: 1000
```

In a real-world programming situation, you'd probably use one of the supported *looping constructs* to iterate through each element of the array, rather than write each line individually as shown above. However, this example clearly illustrates the array syntax. You'll learn about the various looping constructs (`for`, `while`, and `do-while`) in the Control Flow section.

Declaring a Variable to Refer to an Array

The above program declares `anArray` with the following line of code:

```
// declares an array of integers
int[] anArray;
```

Like declarations for variables of other types, an array declaration has two components: the array's type and the array's name. An array's type is written as *`type`*`[]`, where *`type`* is the data type of the contained elements; the square brackets are special symbols indicating that this variable holds an array. The size of the array is not part of its type (which is why the brackets are empty). An array's name can be anything you want, provided that it follows the rules and conventions as previously discussed in the naming section. As with variables of other types, the declaration does not actually create an array — it simply tells the compiler that this variable will hold an array of the specified type.

Similarly, you can declare arrays of other types:

```
byte[] anArrayOfBytes;
short[] anArrayOfShorts;
long[] anArrayOfLongs;
float[] anArrayOfFloats;
double[] anArrayOfDoubles;
boolean[] anArrayOfBooleans;
char[] anArrayOfChars;
String[] anArrayOfStrings;
```

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

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

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

Creating, Initializing, and Accessing an Array

One way to create an array is with the `new` operator. The next statement in the `ArrayDemo` program allocates an array with enough memory for ten integer elements and assigns the array to the `anArray` variable.

```
// create an array of integers
anArray = new int[10];
```

If this statement were missing, the compiler would print an error like the following, and compilation would fail:

```
ArrayDemo.java:4: Variable anArray may not have been initialized.
```

The next few lines assign values to each element of the array:

```
anArray[0] = 100; // initialize first element
anArray[1] = 200; // initialize second element
anArray[2] = 300; // etc.
```

Each array element is accessed by its numerical index:

```
System.out.println("Element 1 at index 0: " + anArray[0]);
System.out.println("Element 2 at index 1: " + anArray[1]);
System.out.println("Element 3 at index 2: " + anArray[2]);
```

Alternatively, you can use the shortcut syntax to create and initialize an array:

```
int[] anArray = {
    100, 200, 300,
    400, 500, 600,
    700, 800, 900, 1000
};
```

Here the length of the array is determined by the number of values provided between *{* and *}*.

You can also declare an array of arrays (also known as a *multidimensional* array) by using two or more sets of square brackets, such as `String[][] names`. Each element, therefore, must be accessed by a corresponding number of index values.

In the Java programming language, a multidimensional array is simply an array whose components are themselves arrays. This is unlike arrays in C or Fortran. A consequence of this is that the rows are allowed to vary in length, as shown in the following `MultiDimArrayDemo`program:

```
class MultiDimArrayDemo {
    public static void main(String[] args) {
        String[][] names = {
            {"Mr. ", "Mrs. ", "Ms. "},
            {"Smith", "Jones"}
        };
        // Mr. Smith
        System.out.println(names[0][0] + names[1][0]);
        // Ms. Jones
        System.out.println(names[0][2] + names[1][1]);
    }
}
```

The output from this program is:

```
Mr. Smith
Ms. Jones
```

Finally, you can use the built-in `length` property to determine the size of any array. The code

```
 System.out.println(anArray.length);
```

will print the array's size to standard output.

Copying Arrays

The `System` class has an `arraycopy` method that you can use to efficiently copy data from one array into another:

```
public static void arraycopy(Object src, int srcPos,
                             Object dest, int destPos, int length)
```

The two `Object` arguments specify the array to copy *from* and the array to copy *to*. The three `int` arguments specify the starting position in the source array, the starting position in the destination array, and the number of array elements to copy.

The following program, `ArrayCopyDemo`, declares an array of `char` elements, spelling the word "decaffeinated". It uses `arraycopy` to copy a subsequence of array components into a second array:

```
class ArrayCopyDemo {
    public static void main(String[] args) {
        char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e',
                            'i', 'n', 'a', 't', 'e', 'd' };
        char[] copyTo = new char[7];

        System.arraycopy(copyFrom, 2, copyTo, 0, 7);
        System.out.println(new String(copyTo));
    }
}
```

The output from this program is:

```
caffein
```

Previous page: Primitive Data Types
Next page: Summary of Variables