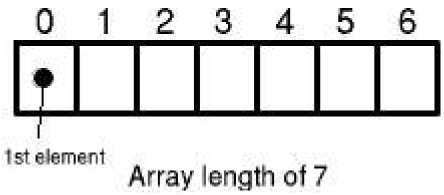
Java Arrays

Source http://java.about.com/od/beginningjava/l/aa_array.htm

Java Arrays

Fixed-Length Data Structures

As we saw in Java Variables, you can store program data in variables. Each variable has an identifier, a type, and a scope. When you have closely related data of the same type and scope, it is often convenient to store it together in a data structure instead of in individual variables. The most common data structure is the array. Arrays are fixed-length structures for storing multiple values of the same type. An array implicitly extends java.lang.Object so an array is an instance of Object. But arrays are directly supported language features. This means that their performance is on par with primitives and that they have a unique syntax that is different than objects.



Structure of Java Arrays

The Java array depicted above has 7 elements. Each element in an array holds a distinct value. In the figure, the number above each element shows that element's index. Elements are always referenced by their indices. The first element is always index 0. This is an important point, so I will repeat it! The first element is always index 0. Given this zero-based numbering, the index of the last element in the array is always the array's length minus one. So in the array pictured above, the last element would have index 6 because 7 minus 1 equals 6.

Java Array Declaration

An array variable is declared the same way that any Java variable is declared. It has a type and a valid Java identifier. The type is the type of the elements contained in the array. The [] notation is used to denote that the variable is an array. Some examples:

```
int[] counts;
String[] names;
int[][] matrix; //this is an array of arrays
```

Java Array Initialization

Once an array variable has been declared, memory can be allocated to it. This is done with the **new** operator, which allocates memory for objects. (Remember arrays are implicit objects.) The **new** operator is followed by the type, and finally, the number of elements to allocate. The number of elements to allocate is placed within the [] operator. Some examples:

```
   counts = new int[5];
   names = new String[100];
   matrix = new int[5][];
```

An alternate shortcut syntax is available for declaring and initializing an array. The length of the array is implicitly defined by the number of elements included within the {}. An example:

```
   String[] flintstones = {"Fred", "Wilma", "Pebbles"};
```

Java Array Usage

To reference an element within an array, use the [] operator. This operator takes an int operand and returns the element at that index. Remember that array indices start with zero, so the first element is referenced with index 0.

```
   int month = months[3]; //get the 4th month (April)
```

In most cases, a program will not know which elements in an array are of interest. Finding the elements that a program wants to manipulate requires that the program loop through the array with the **for** construct and examine each element in the array.

```
    String months[] =
       {"Jan", "Feb", "Mar", "Apr", "May", "Jun",
        "July", "Aug", "Sep", "Oct", "Nov", "Dec"};

   //use the length attribute to get the number
   //of elements in an array
   for(int i = 0; i < months.length; i++ ) {
      System.out.println("month: " + month[i]);
   }
```

Using Java 5.0, the enhanced **for** loop makes this even easier:

```
    String months[] =
       {"Jan", "Feb", "Mar", "Apr", "May", "Jun",
        "July", "Aug", "Sep", "Oct", "Nov", "Dec"};

   // Shortcut syntax loops through array months
   // and assigns the next element to variable month
   // for each pass through the loop
   for(String month: months) {
      System.out.println("month: " + month);
   }
```

Arrays are a great way to store several to many values that are of the same type and that are logically related to one another: lists of invoices, lists of names, lists of Web page hits, etc. But, being fixed-length, if the number of values you are storing is unknown or changes, there are better data structures available in the Java Collections API. These include ArrayList and HashSet, amongst others.

~ ~ ~ End of Article ~ ~ ~