Return to Module Overview Page (https://onlinelearning.berkeley.edu/courses/665040/pages/module-two)

2.6: Arrays

Arrays are a very useful programming tool. A Java array is like a list of variables that all share the same name. You refer to one individual variable by supplying that name plus a number that gives the position in the list:

```
// create an array
int [] myArray = new int[10];
// set the first element to 7
myArray[0] = 7;
// read the fourth element
int i = myArray[3];
```

To create an array in Java, you always use the "new" operator, as shown above. "new" is something akin to malloc() in C; Visual Basic's "DIM" statement is a poor cousin. What any of these statements does is to set aside space for a certain number of items in the list (or "elements in the array," to use the proper terminology).

Any array created this way has a definite number of elements in it. What's more, the array knows how many, and you can ask it:

```
System.out.println("myArray has " +
                   myArray.length +
                   "elements.");
```

prints "myArray has 10 elements." Note that the first one is named myArray[0], and the last one is myArray[9], for a total of 10 elements. Although this may seem strange to you, depending on what languages you already know, it lets you write loops such as

```
for (int i=0; i<10; i++)
    System.out.println("myArray[" + i +
                       "] is " + myArray[i]);
```

without using the uglier >= operator, for example.

If you try to access the eleventh element of myArray:

```
myArray[10] = 0;
```

your programs stops and prints something like

```
java.lang.ArrayIndexOutOfBoundsException:
          at test.main(test.java:5)
```

This is like what happens in Visual Basic, but unlike what happens in C. C has no runtime checks for this kind of thing, so what generally happens is memory gets corrupted and perhaps the program (eventually) crashes.

It is important to realize that the variable myArray is not the array itself; the variable merely refers to (points to) the array. The array itself is an object, a chunk of memory on the freestore. Any number of variables (including 0) may refer to a given array:

```
int[] arrayA = new int[10];
int[] arrayB = arrayA;
```

Both variables refer to the same array. If I now assign a value to arrayA[3], I would read the same value from arrayB[3]. I can make neither of these variables refer to that array:

```
arrayA = null;   arrayB = null;
```

null is a special value in Java which means "no object." Now the array we allocated is unreachable. This is not a memory leak as it would be in C, though. The array is silently recycled by Java; this process is called garbage collection, and we'll discuss it further in a later module.

Note that in Java, if you want to grow an array, the only way to do it is to create a new, bigger array, copy the old data into the new array, and then assign the new one to the old variable:

```
// create an array
int [] temp = new int[20];
System.arraycopy(myArray, 0, temp, 0, myArray.length);
// assign the new array to the old
myArray = temp;
```

The System.arraycopy() function (you can look it up in the online API docs) quickly copies the contents of one array into another. It's much faster than the equivalent "for" loop:

```
for (int i=0; i<myArray.length; i++)
   temp[i] = myArray[i];
```

Notice that assigning directly to the array "myArray" discards the whole list of elements set aside by the original "new" call. The variable myArray now refers to the new storage area allocated by the second "new." Java automatically figures this out and will deallocate the memory if it needs to reuse it.

Note that you can have arrays containing any Java data type, not just integers. For example:

```
String [] s = new String[3];
for (int i=0; i<3; i++)
   s[i] = "foo";
```

You can also initialize arrays using the equivalent shortcut notation:

```
String [] s = { "one", "two", "three" };
```

Note that in Java, multidimensional arrays are actually arrays of arrays. The line

```
int [][] matrix = new int[10][10];
```

allocates a two-dimensional, ten-by-ten array of integers. This one line is exactly equivalent to

```
int [][] matrix = new int[10][];
for (int i=0; i<10; i++)
   matrix[i] = new int[10];
```

You can therefore easily make two-dimensional arrays that are not rectangular:

```
int [][] triangle= new int[10][];
for (int i=0; i<10; i++)
   triangle[i] = new int[i];
```