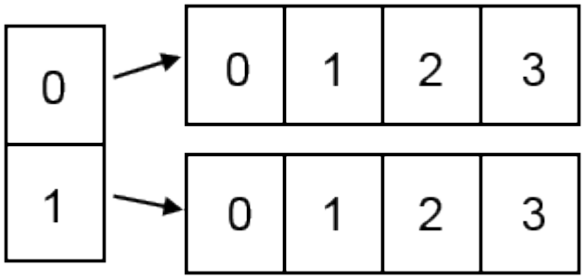
Students studying for the AP Computer Science exam, starting in the Fall of 2014, should append this text at the end of Chapter Thirteen, Lesson One.

Arrays of Arrays

In Java, arrays of 2 or more dimensions are actually stored as arrays of arrays! This means each element in the first dimension's array is itself another full array. Consider this example again:

```
int[][] my2DArray = new int[2][4];
```

The first dimension has 2 elements, so Java will create a 1D array with 2 elements. Then, each of those first two elements themselves is an array of 4 elements, so Java will create two more 4-element arrays. Each of the elements in the first dimension will hold a reference to one of those additional 4-element arrays.



You can access each element easily with double square brackets as we have already shown.

```
int[][] my2DArray = new int[2][4];
int value = my2DArray[0][1];
```

The first index goes with the first dimension (with 2 elements in our example) and the second index goes to the second dimension (with 4 elements). So **my2DArray**[0][1] will start at the 0th element in the first array and then go to the 1st element in the array held in that 0th element. If you view a 2D array as we have drawn with the first dimension elements stacked in rows from top to bottom, this is called **row-major order**. The first index determines the row, and the second index determines the column.

You can also get a reference to one of the 1D arrays and treat it like a normal 1D array.

```
int[][] my2DArray = new int[2][4];
int[] row = my2DArray[1];  // get a reference to the second row
```

Here we declare a normal 1D array of integers called **row**. Then instead of initializing the variable with a new array, we just access the 2D array with a single index to get a reference to the entire 1D array stored in that element!

Once you have that reference, you can do anything with it that you would normally do with a 1D array, including walking it with a **for**() loop, passing the array variable into a function, getting a value out with a single index, etc.

```
int[][] my2DArray = new int[2][4];
int[] row = my2DArray[1];  // get a reference to the second row
int data = row[3]; // get the last value in the second row
```

Arrays of Objects

Arrays can contain not only basic data types, but also references to objects. To declare arrays of objects, simply use the object name instead of the basic data type. Let's pretend an object named **Dog** has been defined, and create an array of 3 **Dog** objects.

```
Dog[] myDogs = new Dog[3];  // declare a 3-element array
myDogs[0] = new Dog();  // store a new Dog object in element 0
myDogs[1] = new Dog();  // store a new Dog object in element 1
myDogs[2] = myDogs[1];  // elements 2 and 1 refer to the same Dog
```

When the array of objects is declared on the first line, each element in the array contains **null** and not a valid **Dog** object. Declaring arrays of objects does not automatically create objects to store in each element! We need to manually create a new **Dog** object to store in each element. Notice on the last line that we actually just copy the reference from **myDogs**[1] to **myDogs**[2]. This means both elements in the cell hold the same reference to the same **Dog** object! So there are only two distinct **Dog** objects created by this example.

