Sixth Lesson: Java Continued — All About Arrays

We hope you're enjoying the final few days of summer break!! This week's lesson is mainly about arrays in Java, for-loops, and 2D arrays!

A quick note before we start: To access arrays and array lists, you have to import the classes into your program. You will most likely be prompted to do so by your IDE.

Arrays

Arrays simply help programs become more efficient, especially in managing large data sets. Unlike Python lists (which you will probably see more often in that language compared to Python arrays), elements in an array must be of the same type—for example, you cannot put a String and an integer in one array.

Thus, we have to declare the type of variables that will go in the array every time we create one. If we wanted to declare and create an array containing 100 int variables, we'd simply code:

```
int[] nums = new int[100];
```

The brackets indicate that you are creating an array and not a single value variable. This is nonnegotiable, but if you want to create an array that will not be filled with integers, you simply have to replace the `int` with another type. For example, an array of strings that is five items long:

```
String[] words = new String[5];
```

You can create arrays of `String`, `Boolean`, `Int`, and even your own objects! It is imperative that you define how many objects you want inside your array; arrays, unlike array lists, won't allow you to change the length of your array after you've made it.
You can also initialize the array with the objects themselves instead of creating an empty array of a certain length and then adding all of your values manually. To do that, you simply have to create a list (remember that in Python lists are created with elements, separated by commas, enveloped by brackets []), but with squiggly brackets {} instead of normal brackets.

```
String[] newWords = {"Peanut butter", "Jelly", "Jam"}
```

Other functions like indexing and finding the length remain similar to Python. A few of the many are listed below.

```
nums[0] = 5; // updates a value at a specific index
System.out.println(words[1]); // accesses the value at an index
int numsLen = nums.length;
int wordsLen = words.length;
nums.add(1); // will add the int 1 to the end of the array
```

For Loops
The syntax for a for-loop in Java is pretty different from Python (it's a lot clunkier, for one), but it's relatively simple and is somewhat similar to the while loop. There are three main components that the two aforementioned loops require:

1. Looping variable declaration / initialization
2. Condition for stopping the loop
3. An update to the variable

The components are enclosed in parentheses and separated by semicolons. A typical loop will look like:

```
for (int i = 0; i < 10; i++){
     // code that gets looped goes in here
}
```

If we combine the two concepts of for-loops and arrays, we can perform operations on the elements of an array.

```
Random rand = new Random();
```

```
int[] numbers = new int[100];

//Randomly assign integer values to each index in numbers
for (int i=0; i<numbers.length; i++) {
     numbers[i] = rand.nextInt(100);
}

//Print out each element in numbers on a separate line
for (int i=0; i<numbers.length; i++) {
     System.out.println(numbers[i]);
}
```

Recall from lesson 3 way back when that we introduced for loops that are a bit more efficient in certain circumstances. (If it wasn't made clear, these "special" for loops are good if you are trying to check an index, but if you want to change that index, the "special" for loop would not work.) Java has something similar!

```
for (int i: nums){
     System.out.println(i);
}
```

To do these for loops, the statement inside the parentheses is much simpler. Here, you have to define what sort of element you're looping through (define the type) and each value inside the nums array is basically saved into the i variable, which will be replaced each time your for loop runs. You would use "i" to do whatever you need to do with the specific array elements inside the for loop, just as you would use nums[i] in the usual for loop.

2D Arrays

Unsurprisingly, you can also store items in arrays that have both rows and columns. These are useful if data are naturally organized in this way, like in pictures! Java stores 2D arrays as arrays of arrays. This means that in each element of the "outer" array, the "inner" array gets referenced.

The "outer" and "inner" arrays are dependent on the orders. The **row-major** order means that when storing the data, the first row is followed by the subsequent rows. In the **column-major** order, the data for the first column is stored and followed by subsequent columns. For our purposes, we'll be using the row-major order.

In order to declare a 2D array, we have to specify the type of elements stored, and then use `[][]` to indicate that it is a 2D array.

```
int [][] values;
String[][] names;
// These only declare the array, but do not actually create it.
```

To instantiate an array, we use the `new` keyword, followed by the variable type and the numbers of rows and columns, similar to how we did it for arrays except you have to define the number of rows along with the number of columns.

```
values = new int [5][2];
names = new String [2][5];
```

Indexing also applies to 2D arrays. If we wanted to initialize the array elements:

```
values[0][0] = 10; // sets the value in the first row and first
                   // column to 10
```

The number of elements in a 2D array is the number of rows multiplied by the number of columns. To get the length (or the size) of rows or columns:

```
values.length // returns the number of rows
values[0].length // returns the number of columns
```

The reason why `values.length` returns the number of rows and `values[0].length` returns the number of columns is because we are using the row-major order. This means that rows, being major, is the thing that is accessed first and more broadly. Columns, being minor, require more specification to be accessed.

Looping through a 2D array will look a little something like this:

```
for (int i = 0; i < values.length; i++){
     for (int j = 0; j < values[0].length; i++){
          // code for each specific element goes here
     }
}
```

The nested for loops is because you have rows and columns. The first for loop will access the rows, then the second for loop will go in and access each column, giving you individual values.

Array Lists

Array lists are pretty handy because unlike arrays, their size can be modified. This means that they can be as long as you want (within limits, obviously), and that you don't have to create a new array every time you want to change its size. They are similar to Python lists in this respect.

To create and instantiate an array list, you would do something like this:

```
ArrayList<String> strlist = new ArrayList<String>();
```

The purpose of the < > is to declare what elements the array lists will hold, since array lists, like arrays, can only hold one type of variables. There are a few differences between the keywords you would use to indicate this in an array and an array list. For example, if you wanted to create an array list of integers, you would do this:

```
ArrayList<Integer> intlist = new ArrayList<Integer>();
```

The reason for the use of `Integer` instead of `int` is that you cannot use primitive types (e.g. int, boolean, char) for array lists since they can only hold objects. Instead, you would have to use the corresponding class (e.g. Integer, Boolean, Character), an equivalent, in its place.

When first created, array lists are initially empty. To add items to an array list, you would just use the `.add()` method, similar to how we did it with arrays. To get the length of an array list, you have to use `.size()`, which is different from an array and a string (arrays have `.length` while strings have `.length()` with the parentheses). To get an element of an array list, you use the method `.get(i)`; brackets won't work here.

To iterate through an array list, you can use a for loop like you would for an array. Here's an example:

```
ArrayList<String> jams = new ArrayList<String>();
for (int i = 0; i < jams.size(); i++){
     System.out.print(jams.get(i) + " ");
}
```

Summary

Today we covered 1-D and 2-D arrays, along with for loops and array lists. Both arrays and array lists store lists of objects (which must be defined and there can only be one), but array lists do not have a defined size and employ a couple different methods. 2-D arrays have both columns and rows, making them a little more tricky. And for loops, well, we've covered them before!