Lesson 5: Arrays for Days

5.1 Introduction:

The world is filled with countless examples of relationships. A spoon is related to a fork and knife because they are all pieces of cutlery. You are related to your family. Because computer programs are intended to handle real world situations, we often find these same relationships manifested in code.

In Lesson 3 we covered the topic of variables. These can be used to store single useful data points, such as a name or an age. In today's lesson we will cover the most common type of *Collection* the array. Collections are used to group together related variables in a meaningful way. There are many different types of collections, and each are tailored to do something slightly different than other collections. Today we will look at the most common Collection, the *Array*.

5.2 Arrays Explained

Assuming you, like most humans, eat food, the concept of a grocery list should be familiar. A piece of paper sits on the counter and you write down everything that is missing from the fridge. For me, this typically looks something like this:

"Ramen"
"Water"
"Coffee".

Maybe your list is a little different, but the important part is the structure. Here, I have three names of foods in a list. Using your knowledge of Lesson 3, how would you put this information in a computer program?

```
String itemOne = "Ramen";
String itemTwo = "Water";
String itemThree = "Coffee";
```

This is certainly an option, but it doesn't show any relationship between the list items. For that, we use an array:

```
String[] shoppingList = new String[3];
shoppingList[0] = "Ramen";
shoppingList[1] = "Water";
shoppingList[2] = "Coffee";
```

This might look ugly but let's go through these four lines of code one step at a time.

The first line of code is very similar to what we looked at with variables. This is how we initialize and declare an array. To the left of the equals sign we see:

```
String[] shoppingList
```

String[] is the variable type, note that this is different from String. The square brackets denote that this is an array of String variables. shoppingList is the name of the variable.

To the right of the equals sign we see:

```
new String[3];
```

We will cover exactly what this means and why it is necessary in a later lesson, but for now the important point is that the number inside this square bracket is the size of the array. We are telling the computer that we want a new array of Strings and we want to put 3 Strings in that array.

The next three lines of code actually put the Strings in the array.:

```
shoppingList[0] = "Ramen";
shoppingList[1] = "Water";
shoppingList[2] = "Coffee";
```

These three lines all have a set of square brackets, again indicating that shoppingList is an array, and a number inside. This number is called the index and allows the computer to interact with specific elements of the array. Notice how the first element, "Ramen", has an index of 0. The first element of a java array always uses the index 0. The first line of code assigns the value "Ramen" to the variable shoppingList[0], and so forth. You can think of shoppingList[number] as a String while the entirety of shoppingList is an array.

5.2 Declaring and Initializing:

In the previous section we saw one way to declare and initialize an array. This is the most common way to declare an array and is a syntactical construction that you will see throughout the Java language. However, there are two more useful ways to initialize an array. Let's create an array of the integers from 0 to 9, inclusive.

```
int[] zeroToNine = {0,1,2,3,4,5,6,7,8,9};
```

Notice how declaring an array (the code to the left of the equals) is the exact same as above. To the right of the equals sign, we have a set. This set must be of the same type as the array, in this case integer, is enclosed in curly brackets and a comma separates each element. The set is essentially a simplified version of this:

```
zeroToNine[0] = 0;
zeroToNine[1] = 1;
zeroToNine[2] = 2;
zeroToNine[3] = 3;
zeroToNine[4] = 4;
zeroToNine[5] = 5;
zeroToNine[6] = 6;
zeroToNine[7] = 7;
zeroToNine[8] = 8;
zeroToNine[9] = 9;
```

As you can see, initializing with a set greatly simplifies the process of assigning values to the array. It does have limitations however. Using this strategy requires the programmer to know exactly what he wants the values in the array to be. Let's look at something a little bit more complicated. Let's say we want an array filled with the numbers 0-9 cubed. That is, the first number will be 0*0*0, the second 1*1*1, and the third 2*2*2 etc. We can use a nifty combination of our basic arithmetic, loops and arrays to achieve this.

```
int cubes[] = new int[10];
for (int i = 0; i < 10; i++)
{
    cubes[i] = i*i*i;
}
```

Before reading on, try to figure out what's going on here.

In the first line we declare an array of integers, and say that it will be of size 10. We then create a for-loop structure that begins at 0, increases by 1 with each iteration, and stops when the condition "i<10" is false. During each iteration of the loop, we assign the value of i*i*I to the element of cubes located at index i.

5.3 Updating:

Like other variables, an array is not required to stay constant throughout our program. It can be updated and manipulated freely. Fortunately, it is extremely simple to update the contents of an array. Using the assignment operator (equals sign) you can simply give an element a new value, without needing to delete the previous value or anything like that. Imagine we have entered a strange new world where cubing a number is the exact same as the number system on Earth, except cubing the number 9. 9 cubed is 7 in this strange new world. After the for loop above, we can simply add the line:

```
cubes[9] = 7;
```

This element will originally be set in the for-loop, but then we will re-set it to 7 afterwards.