9.4 Passing Complex Data

We specifically said at the end of **section 9.3 (https://canvas.swansea.ac.uk/courses/44525/pages/9-dot-3-methods-that-dont-return-anything-and-how-parameters-are-passed)** that you cannot change the value of parameters outside methods, and we showed an example. We also said that things were not so simple with more complex data. Here's an example:

```
import java.util.ArrayList;
import java.util.Arrays;

public class ChangeArrayList {
     public static void main(String[] args) {
          //Create an ArrayList quickly from an array
          ArrayList<Integer> list =
               new ArrayList<>(Arrays.asList(1, 2, 3));

          //Call method to add 1 to all items in the list
          addOne(list);

          //Print out the list using for-each
          for(int val : list) {
               System.out.println(val);
          }
     }

     //Method to add 1 to elements of an ArrayList
     static void addOne(ArrayList<Integer> valList) {
          for(int i = 0; i < valList.size(); i++) {
               valList.set(i, valList.get(i) + 1);
          }
     }
}
```

(As an aside, notice if you want an `ArrayList` that contains integers you need to write `Integer` not `int` - for reasons beyond this module.)

If you run this you'll find the output is 2, 3, 4 – *but we put 1, 2, 3 in the ArrayList* – ***so obviously we've managed to change it. How?? We said this was not possible!***

Here's a contradictory example:

```
import java.util.ArrayList;
import java.util.Arrays;

public class ChangeArrayList {
     public static void main(String[] args) {

          //Create the list
          ArrayList<Integer> list =
               new ArrayList<>(Arrays.asList(1, 2, 3));

          //Call method to delete it completely
          zapArrayList(list);
```

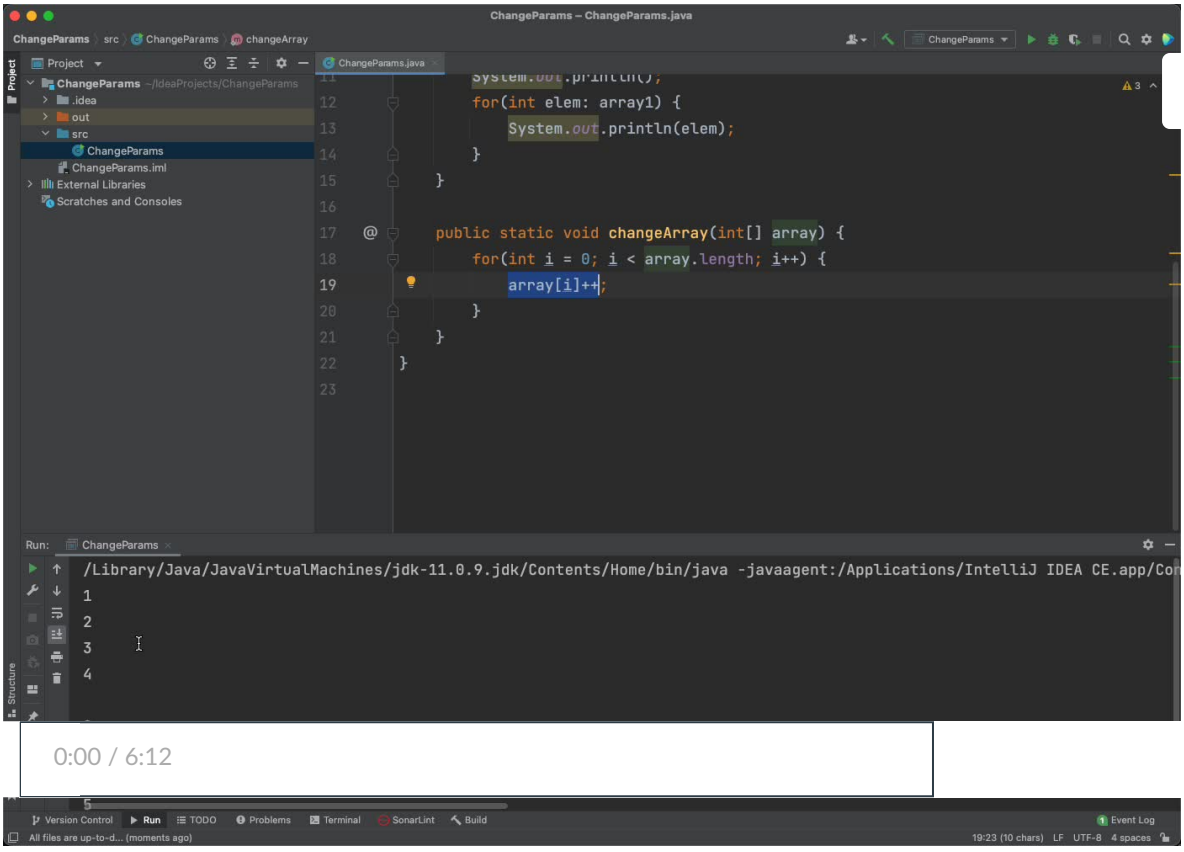
```
            //Print it out
            for(int val : list) {
                  System.out.println(val);
            }

      }

      //Method to set ArrayList to null to delete it
      static void zapArrayList(ArrayList<Integer> valList){
            valList = null;
      }
}
```

This one has a method that *explicitly* sets the `ArrayList` passed in as an argument to *null* – i.e. it basically *deletes it*. And yet, after the method returns the contents of the `ArrayList` are still present. ***So this one does NOT change it's argument - what is going on?***

So what is going on? You can see this example, and an explanation in the video below - using a slightly different example - *but make sure you also read and understand the text following it:*



The Answer? Complex Objects are *References*

If you didn't read it the first time, now go back to **Chapter 4 Section 4.1 (https://canvas.swansea.ac.uk/courses/44525/pages/4-dot-2-strings-and-equals-danger)** , and read the section **Strings and Equals Danger: Skip on First Reading if New to Programming** and especially look at **Figure 1.** Here's a version of that picture that will help:

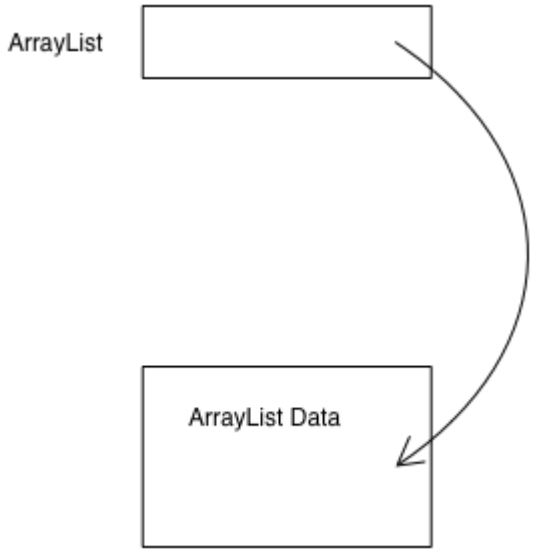


Figure 1: References to Data

Complex data like an `ArrayList` is *not* stored in the same way as simple data. Instead the actual variable is a *reference* to the real data, which is *elsewhere* in memory. This is done for a perfectly good reason, which I'm happy to explain to anyone who wants to know, but is a bit off topic here. What this means is that *the reference is the value that is passed* – it *cannot* be changed (which why `zapArrayList` didn't work) *but the data it refers to* ***can*** *be changed* (which is why `addOne` *did* work). Here's another picture to help:

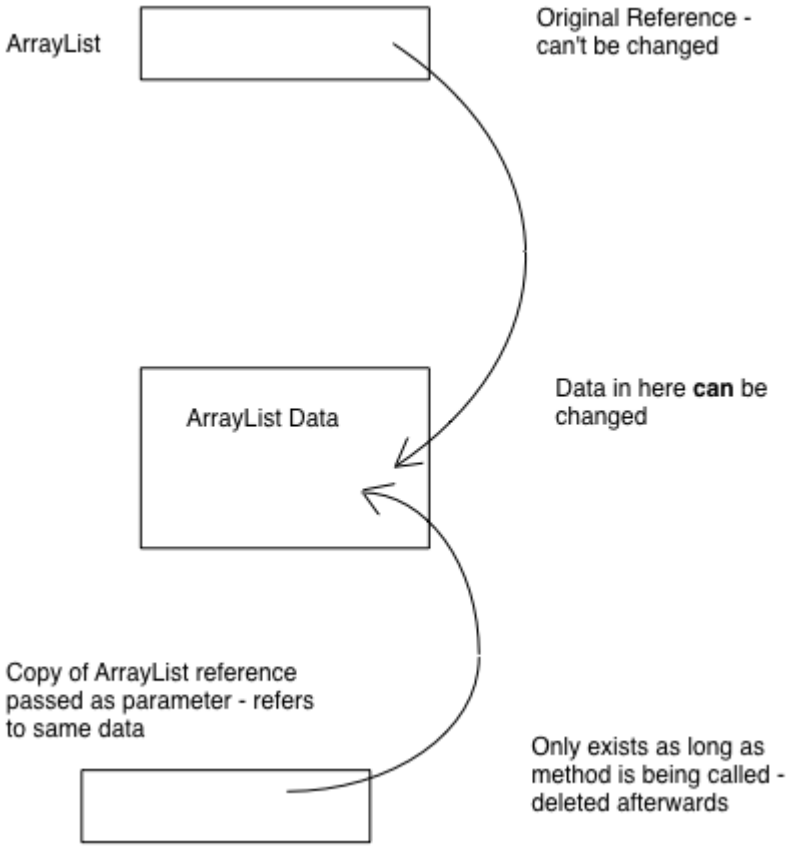


Figure 2: Original Reference and Parameter

When we pass something like an `ArrayList` as a parameter, we make a *copy of the reference*, and *pass that reference* – but *the reference still points to the same data*. Whatever changes we make to *the copy of the reference* (by, say, setting it to null) get discarded when the method ends. But we *can* make changes to the *actual data* that the reference copy 'points' to.