The contents of this document represent and go over the topics covered in lecture on 3/21 and 3/23 Use this to review the content as you please, also be sure not to use these in place of lecture as these documents will only cover lecture highlights and important details and do not represent the full scope of what you may be tested on. As always if you have any questions ask on ED!

Lecture 17 - 3/21/2023

We introduced a few new concepts that you need to know how to read and identify but not necessarily know the anatomy of. The first was us converting a value to its currency form. We used NumberFormat and Locale classes to do this, consider the following line from the codio file:

```
NumberFormat fmt= NumberFormat.getCurrencyInstance(Locale.US);
```

We use the static method of the NumberFormat class to get the formatting for our desired currency, we use a Locale object as the argument to the method. We can either create a Locale object as described in the comment in the file or we can use a constant in the Locale class that matches our desired currency format which is what the above line does.

We then use a static method from the String class called format to format strings, which takes at least 2 arguments: a string to format, and a variable to replace our formatting placeholder. We then can use \t to add a tab \n for a new line and %xsv or %-xs to pad the left or right of the string respectively.

Finally we revisited the idea of Object references and what it means to count Object references, and what happens when we have an Object in memory that has no reference to it at some point in the program. Take the following code snippet:

```
-------------------------------------------------------------
AudioBook[] temp = new AudioBook[collection.length*2];

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

collection = temp;
-------------------------------------------------------------
```

We start prior to entering this snippet with a single array called collection that holds references to AudioBooks. Once we enter the snippet we now have two arrays: collection and temp, with temp having twice the length of collection, after the for-loop runs we know the first [0,collection.length-1] indices in temp point to valid AudioBooks - the same ones in the same respective indices

as collection. From [collection.length-collection.length*2-1] they are all references to null. When we reach the final line we switch the object that the collection reference was pointing to, so that it matches the object that temp points to. There are now two references to the same object in memory and now the original object collection pointed to has no references pointing to it and thus is garbage and the memory can be reclaimed by the JVM. Once we leave the code snippet temp goes out of scope and since we still have a reference to the object temp pointed to there is no memory to reclaim but now we are back to having a single array called collection that now has twice the length.

Lecture 18 - 3/23/2023

This lecture is mainly focused on ArrayLists. Which are essentially mutable arrays since we can dynamically change the size of the ArrayList. In order to be able to use ArrayLists, we must import the necessary prerequisite package. We do so through the following statement:

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

Once we have wrote the necessary import we can declare and initialize an ArrayList as follows:

```
ArrayList<ObjectType> nameOfArrayList = new ArrayList<[optionalObjectType]>();
```

When it comes to actual Java code the above generic form can be seen as one of the following:

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

Both of these are acceptable ways to declare and initialize an ArrayList, the ObjectType is optional on the right hand side of the equal sign.

ArrayLists are very method intensive, unlike the default implementation of Arrays, in order to add elements, remove elements, access elements, and set elements are very different then the same things when dealing with Arrays.

To add an element to an ArrayList we can use the add(E element) or add(int idx, E element) methods

The first would be used to add an element to the end of the ArrayList you are using as the implicit parameter. The second would be used to add an element to a specific index and it shifts the other elements to the right by one as a result. Some examples of Java would be as follows:

```
nameOfArray.add(5);
nameOfArray.add(0, 7);
```

Use the above descriptions to determine which does which and what exactly it does
In order to remove elements from an ArrayList we can use remove(int idx) or remove(E element) to do this, the first removes an element from a specific index and the second removes the first instance of element in ArrayList assuming it exists.

To set an element of an ArrayList we can use the set(int idx, E element) method. It takes an int for the index and then whatever element we wish to set. It simply replaces the value at position idx.

To access an element of an ArrayList we can use the get(int idx) method which returns the element at position idx.

Putting all of this together we can iterate through our ArrayList and print out all the elements:

```
for(int i = 0; i < nameOfArray.size(); i++){
  System.out.println(nameOfArray.get(i));
 }
```

Notice that rather than .length like we did for Arrays, for ArrayLists since the size of the ArrayList is dynamic we use the method .size().