Lecture 17 - 3/19/2024

We went over the Comparable interface again in order to properly understand it in the context of our AudioBook example.

To refresh, the following is the method header for the compareTo method when dealing with AudioBooks:

```
public int compareTo(AudioBook other)
```

We need to provide an implementation for the compareTo method whenever we implement the Comparable interface. To implement the Comparable interface we need to include the following in our class header:

```
implements Comparable<AudioBook>
```

If you were implementing this for another class then you would replace AudioBook with whatever class you are implementing it for. When it comes to implementing the compareTo method it is convention to return 1 when the calling object comes after the argument, 0 if it is equivalent, and -1 when the calling object comes before the argument. A proper call to compareTo with two AudioBook objects a and b would look like:

```
a.compareTo(b)
   OR
b.compareTo(a)
```

The specific order you do it in makes a difference in how to interpret your results. As mentioned in lecture, you need not bind yourself to previously mentioned convention but you DO need to bind yourself to the convention of positive, negative, and zero for their respective cases above.

Moving on from this we discussed ArrayLists in Java, these are more similar to lists in a language like Python. ArrayLists are still homogeneous collections of items but unlike arrays they are not fixed in length. The length of an ArrayList is always just enough for us to make use of. In order to make use of

ArrayLists in Java is we need to import them from the util package as follows:

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

To instantiate a new ArrayList we do it in one of the following ways:

```
ArrayList<ClassName> variableName = new ArrayList<ClassName>();
```

OR

```
ArrayList<ClassName> variableName = new ArrayList<>();
```

Two key things to note about this:

1. The second usage of ClassName is optional
2. You cannot make an ArrayList of primitives; you must use their wrapper object types instead.

Here are the wrapper types for all the primitive types:

1. Integer
2. Double
3. Character
4. Byte
5. Short
6. Long
7. Float
8. Boolean

Fortunately Java doesn't make this your problem as both of the following are valid:

```
int x = IntegerArrayList.get(0);
Integer x = IntegerArrayList.get(0);
```

Doing the former would suggest that Java unwraps the primitive from the wrapper type. Which is exactly what it does, and it

goes both ways; you do not need to explicitly make Integer objects to add them to the ArrayList.

When working with ArrayLists there are a few key methods that you should be aware of:

1. get(int index) - returns the item at position index
2. set(int index, T value) - replaces the item at position index with the specified value
3. add(T value) - insert value to the end of the list
4. add(int index, T value) - insert value at position index shifts the rest to the right by 1
5. remove(int index) - remove the item at position index
6. remove(T value) - remove the first instance of value
7. size() - returns how many elements are in the ArrayList