COSC 222 – Lab 2: BucketSort Revisited

This lab serves as an intro to Java Interfaces and an Object-Oriented design strategy towards implementing data structures. We will be using `Entry.java` objects which are key,value pairs, with `key` being an integer and `value` being a generic parameter. We will be sorting these Entries according to their key value. The file `Entry.java` is given to you and does not require any modification.

As you should know by now, bucket sort works by placing items into buckets and each bucket is sorted on its own. Once all items are placed, the bucket contents are extracted in a sequence from smallest bucket to largest. In this lab, we will have a `SortedBucket` object which takes an item and stores it in sorted order. We will also be using a `BucketList` object which creates a list of `SortedBucket` objects, and decides which bucket each number (Entry) goes to.

Start with looking at the two Abstract Data Types:

SortedBucketADT.java: an interface that describes what functionality a bucket has. Aside from a constructor and a `toString()` method, these can `.add(Entry)` and will place the Entry in sorted order within the bucket, and they also have a `getBucketContents()` method which returns the contents of the bucket.

BucketListADT.java: an interface that describes the collection of buckets. The constructor takes 3 parameters: `int min, int max, int n`, which represent the expected minimum of Entry keys, expected maximum of Entry keys, and the number of buckets to create. It supports methods `.add(Entry)` and `.addAll(Entries)` to add one Entry or to add many Entries. The .add( ) method should decide which bucket the Entry gets placed in. It also has a way to extract the sorted order of Entries with a `.getSortedOrder()` method.

You do not need to modify these ADT files. Your task will be in implementing and testing them. Namely, `SortedBucket.java` will implement `SortedBucketADT.java` and `BucketList.java` will implement `BucketListADT.java`. These are partially implemented already, and contain //TODO items throughout the file to guide you in the implementation process.

Further, there is a `MainClass.java` present only to show you some sample intended usage of the data structure and sorting method. You can modify or delete this file as you please. Finally, there is a `Test.java` class file which illustrates how one can test the contents of each bucket after adding elements to the bucket list. Once again, there are //TODO items throughout this test file with more specific instructions.

Scoring: Up to 6 points for completing the implementation and up to 4 points for the additional testing you are to include in the test file.

SUBMIT: your edited .java files and a screenshot of your coverage report that shows that your tests provide at least 80% coverage of the project files. You may alternatively show your lab instructor (in lab time) your coverage report and your test cases. Note: Canvas will **not** accept an archive .zip file for this lab.