Fall '14 CIS 212 Assignment 6 – 110/100 points possible – Due Wednesday, 11-19, 11:59 PM

The goal of this assignment is provide exposure to building a new generic data structure. Specifically, this project will involve building a new Set structure that maintains the "add" counts of its unique values so that they can be iterated in sorted order with respect to how many times they've been added to the set. The Set implementation will be optimized for efficient add/remove/contains over efficient iteration, based on the assumption that we'll generally be modifying the structure more often than accessing its values in sorted order.

1. [90] Create a new class OccurrenceSet<T> that implements Set<T>. All methods should function as specified in the Set documentation. Additionally:

- (20) The *add* and *addAll* methods will need to keep track of how many times a value has been added to the set. We are optimizing for efficient add, so *add* should be *O(1)* and *addAll* should be *O(n)* for *n* added values. Hint: see HashMap.
- (20) The *remove*, *removeAll*, and *retainAll* methods should remove the necessary values from the set completely (i.e., not just decrement their counts). We are optimizing for efficient remove, so *remove* should be *O(1)* and *removeAll* should be *O(n)* for *n* removed values.
- (10) The *contains* and *containsAll* methods should behave as documented and operate in *O(1)* and *O(n)* time (i.e., for *n* query values), respectively.
- (10) The s*ize* method should return the number of unique values currently in the set (i.e., not considering "add" counts). This method should be *O(1)*. The *clear* and *isEmpty* methods should behave as documented.
- (20) The *iterator* and *toArray* methods should return an Iterator or array, respectively, with elements sorted by their "add" counts in descending order. We are optimizing for efficient add/remove/contains over iteration, but these methods should still be *O(n lg n)*. The Iterator *remove* method does not need to remove the element from the set (see extra credit). Hint: Creating a new List and using the Collections *sort* method is reasonable here.
- (10) Add a *toString* method that prints the elements in the list in sorted order (i.e., descending with respect to their "add" counts). This method should be *O(n lg n)*.

2. [10] Create a Main class that creates a few OccurrenceSets of various types to test the functionality of your new data structure. For example, a test like:

```
OccuranceSet<Integer> intSet = new OccuranceSet<Integer>();

intSet.add(1);

intSet.add(3);
intSet.add(5);
intSet.add(5);
intSet.add(3);
intSet.add(3);
intSet.add(3);
System.out.println(intSet);

OccuranceSet<String> stringSet = new OccuranceSet<String>();
stringSet.add("hello");
stringSet.add("hello");
stringSet.add("world");
stringSet.add("world");
stringSet.add("world");
stringSet.add("here");
stringSet.add("I");
stringSet.add("am");
System.out.println(stringSet);
```

Should have the output:

```
[3, 5, 1]
[world, hello, am, here, I]
```

You've now got a data structure which will allow you to easily do things like sort words with respect to word counts without ever needing to write this code again (in Java, at least)! Yay! ☺

3. [+10] (Extra credit) Implement your own Iterator class to be returned by the *iterator* method above. Implement the *remove* method of the Iterator such that the current value is removed from the set.

Zip the Assignment6 folder in your Eclipse workspace directory and upload the .zip file to Blackboard (see Assignment 6 link in the Assignments area).