CSSE 220—OBJECT-ORIENTED SOFTWARE DEVELOPMENT

FINAL EXAM, WINTER 2007-08, COMPUTER PART

- These instructions have been slightly modified for their use as a "warm-up assignment for CSSE 230.
- On any of these problems, if you have spent more than a few minutes since you last made progress, you may want to get help.

GETTING AND EXAMINING THE STARTING CODE

I placed an Eclipse project called 220**FinalExam** in your individual SVN repository . It contains partial definitions of the classes that you must enhance, along with the JUnit tests that your code needs to pass for each of the three problems.

You should check out this project now, and commit it often.

You must not change the files that contain the JUnit tests, or the other files that are marked "Do not change". We will copy our (original) versions of these files into a copy of your project folder before testing your code project. Thus the only files you should change are **Anagram.java**, **SortedLinkedList.java**, and **PQ.java**.

To run the JUnit tests for a particular problem in Eclipse, right-click the **...Tests.java** file, and choose Run As … JUnit Test.

SUBMISSION

When you have finished each problem, commit the project back to your repository. Be sure that any of the three files mentioned above that you have changed are checked when you do the Team→Commit in Eclipse.

THE PROBLEMS

I suggest that you read all three problems before you begin solving any of them, so you can plan to make most effective use of your time.

1 .(20 points) **Anagrams.** Two strings are *anagrams* if the characters in one string are an exact rearrangement (permutation) of the characters in the other String. In this problem, we extend the definition to allow for different cases of letters, so that "Spot" and "Tops" should be considered to be anagrams. In the Anagram.java file, I provided the stub for a method, `isAnagram()`, which takes two strings and determines whether they are anagrams. You should complete this method. When it is working, it should pass all of the unit tests in **AnagramTests.java**.

2. (20 points) **SortedLinkedList.** The code I gave you in **Iterator.java**, **LinkedList.java**, and **ListNode.java** is slightly modified from the Linked List (with header node)code for myExam2 solution. I changed the code to require that the elements of the linked list implement the `Comparable` interface; otherwise it is the same as before.

I provided the outline of the definition of a new class called `SortedLinkedList`. As the name implies, the elements of such a list are kept in increasing order. Comments in the **SortedLinkedList .java** code briefly explain what each method is supposed to do/return. Looking at the unit tests may also help clarify what the methods are supposed to do.

As you write and debug each method, run the unit tests. When your code passes all of my unit tests in **SortedLinkedListTests.java**, you will get full credit.

Note that running the unit tests prints a number in Eclipse's console. That is the number of points you have earned so far. This is true for all three programming problems. **Exception:** You must not "Customize" your code so that it gives points only for the specific test cases that I give you. We will look for that in your code.

3. (20 points) **Priority Queue.** The relevant files are **PQ.java**, **PQElement.java**, and **PQTests.java**. As described on page 241 of Weiss, one possible interface to a Priority Queue is via the three methods `insert`, `findMin`, and `deleteMin`. The method call `insert(pri, element)` inserts **element** into this PQ with priority **pri**. Calling `findMin()` returns the element with minimum priority, and `deleteMin()` deletes that lowest-priority element. If two elements have the same priority, the ordering of the elements themselves is used to "break the tie" when determining which is the minimum element of the PQ.

It is quite simple to implement a Priority Queue by using a TreeSet (In CSSE 230, we will learn about a more efficient approach, the Binary Heap). In order to facilitate your work, I have provided a class called PQElement. A **PQ**'s TreeSet member will contain objects of type PQElement. Your job is to fill in the details of the constructor and the three methods of the **PQ** class**.** Your code must pass the unit tests in **PQTests.java**.

After you get the code working (or even if you don't get it working), **answer the following questions** (place your answers in the comment at the beginning of the **PQ.java** file).

a) If we were naïve, we might write the `compareTo` method of the **PQElement** class as

```
public int compareTo(PQElement<T> other){
   return this.priority - other.priority;
}
```

What would go wrong if we do this?

b) I was not quite that naïve, but at first I wrote the `compareTo` method of the **PQElement** class as

```
public int compareTo(PQElement<T> other){
   if (this.priority == other.priority)
      return 1;
   return this.priority - other.priority;
}
```

What would go wrong if we do this?