CSSE 220—OBJECT-ORIENTED SOFTWARE DEVELOPMENT

FINAL EXAM, WINTER 2007-08, COMPUTER PART

The computer part of the exam is open book, open notes, open computer. It consists of 1 problems on 1 page.

- You must disable all communication tools (email, IM, ICQ, IRC, etc.) before the exam starts.
- **Any communication with anyone other than an exam proctor during the exam could result in a failing grade for the course. You may not use headphones or earphones.**
- You may refer to programs you have written for this course—both those assigned and those you have written for practice. You may not, however, refer to programs written by others except those provided in your textbook or by the course staff.
- Regarding materials on the web, you may **only** use the CSSE220 ANGEL web site and pages linked from it, such as the course web pages, and java.sun.com. Also pages that you have previously linked to.
- No search engine use allowed! Also no headphones or earphones.

GETTING AND EXAMINING THE STARTING CODE

I placed an Eclipse project called **FinalExam** in your individual SVN repository . It contains partial definitions of the classes that you need to enhance for this exam, along with 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**.

SUBMISSION

Commit all of your code back to your repository. Be sure that all of the three files mentioned above are checked when you do the Team→Commit in Eclipse. Just in case there is an SVN server problem before your code gets graded, you should also place a ZIP or RAR archive of your entire project on AFS in your turnin folder (in the **Final** folder that I put there).

YOUR QUESTIONS

If you are having trouble with understanding what I am asking for, with subversion, with project management in Eclipse, or with JUnit interface issues, you may ask me or one of the student assistants. For dealing with compile or runtime errors, you are on your own.

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?

CAUTION: Budget your time. If your code passes most of the unit tests for one of the methods, and you can't quickly get the last couple of tests to work, you may want to go on to other methods and come back to the one you're stuck on if you have time later.