COLBY COLLEGE
Department of Mathematics
and Computer Science

CS231 **Allen B. Downey**
Fall 1998

Assignment 3: Priority Queues and Abstract Classes

Reading: Standish Chapter 5.
Due: Monday 5 October
The purpose of this assignment is to test the notion of an ADT by replacing one implementation with another, and to create a generic container using abstract classes.

Pick up the array implementation

I have written an implementation of a Priority Queue using unsorted arrays (actually, Standish wrote it and I modified it a bit). It is available from my web page at:
`http://rocky.colby.edu/cs231/templates/PriorityQueue1/`

1. Create a new CodeWarrior project named PriorityQueue. Pick up the programs (PriorityQueue.java and PQItem.java) and add them to the project, following the same instructions you used to set up the List project.
2. If you are using another Java IDE, do whatever you have to do to compile and run this program.

   NOTE: To keep the number of files small, I have included some code in PriorityQueue for testing. In general, though, the code that uses an ADT (the client) will be in a different module than the implementation. For now, just try to keep them separate in your mind.

Switch implementations

1. Rewrite each of the four methods (the constructor, `size`, `insert`, and `remove`) so that they use a Vector instead of an array.

   The most complicated thing here is that in order to add something to a Vector, you should typecast it to be an Object (although you don't have to). When you get it back from the Vector, you have to typecast it back to a PQItem.
2. Test your program using the client code in `main`.
3. Format your code nicely, including comments, and print out just the four methods that you modified.

NOTE: One of the nice things about ADTs is that you should be able to change implementations like this without modifying the client code. Thus, you should not have to make any changes in `main`.

In order to use the Vector class, you have to import the package that contains it. Add `import java.util.*;` at the beginning of PriorityQueue.java (before the class definition).

Make a generic Priority Queue

So far, our priority queue can only contain one kind of object, PQItem. We would like to be able to make a generic Priority Queue that can contain any kind of item. There are several ways to do this in Java, but one way is:

- Create an abstract class (`interface`) that defines the requirements for objects that want to be put into Priority Queues. In this case, the requirement is that the objects be comparable, so we'll call the abstract class `Comparable`. The book calls it `ComparisonKey`.
- Rewrite Priority Queue so that it can deal with any object that meets the specification of the abstract class; in other words, any object that implements the `Comparable` interface.
- Create a concrete class (`PQItem`) that implements the `Comparable` interface.

Here are the detailed steps you should follow:

1. Make a copy of your working implementation of Priority Queue so that if things go horribly wrong you can back out.
2. Create a new file called Comparable.java and type in the interface specification on page 140 of the text (except call it Comparable rather than ComparisonKey).
3. In PriorityQueue.java, search and replace `PQItem` with `Comparable`, *except in main*.
4. In PQItem, add `implements Comparable` to the first line (following the example on page 141 of the text). Also, modify `compareTo` so that the second argument is a `Comparable` rather than a `PQItem`. That means you will have to do some typecasting inside the method.
5. Finally, in `main` you will have to do some typecasting in order to add items to the Priority Queue and remove them. Since `Comparable` is an abstract class you cannot create instances of it; rather, you have to create `PQItems` and cast them to be `Comparable` before you insert them, then cast them back after you remove them.
6. Print your modified version of PriorityQueue.java and PQItem.java. You don't have to include Comparable.java, but you can if you want. Please take some time to assemble your printouts into a coherent document, so that it is clear that there are two separate programs. Using a highlighter, mark all the sections of code that are different from the previous version (or underline them in pen). If you make life easier for the grader, the grader will make life easier for you!

A little thinking

In the array implementation of the Priority Queue, what happens if you add two items with the same key? Which one of them gets removed first. What about the Vector implementation? Can you tell whether the two implementations will yield the same behavior in this regard?

What does that imply about the ADT specification? I mean, I've been claiming that changing implementations shouldn't affect the client, but it seems like in this case the change of implementation might change the behavior. Is that a problem?

Please answer this question in 2-3 thoughtful paragraphs, and include them with your solutions.