Programming assignment #4: CSE 12 Spring 2013

Here's a description of the fourth programming assignment for CSE 12, Spring 2013. In this assignment, you will implement a priority queue and in turn use it to implement the heapsort algorithm. Along the way you will get more experience with implementing Java interfaces and testing your implementations of them.

>>> Due Fri May 24 5:00 pm

>>> Required solution files: Heap12Tester.java Heap12.java

This means that the files that contain your solution to the assignment must be turned in by the due deadline by using the bundleP4 script while logged into your cs12 account on ieng6. The turnin procedure as for previous assignments, and is described below in the section "Turning in your assignment".

It's a good idea to plan to do the assignment well before the deadlines; terminals and tutors can get very busy at the last minute. As always, keep in mind that in doing this assignment, it's your responsibility to understand and follow the course rules for integrity of scholarship.

Getting started

Review lecture notes on priority queues, heaps, and heapsort. This assignment uses material introduced and discussed there. Create a directory named "P4" under your cs12 home directory to use for developing your solution to this assignment. Your solution to this assignment should be turned in from that directory by the due date and time. You will use the bundleP4 script to do that; see below.

Copy the file `PQueue.java` from the `~/../public/P4/` directory to your P4 directory. This file defines the interface that you will be implementing in this assignment.

Heap12Tester.java

Read the Javadoc documentation on the `PQueue<E extends Comparable<? super E>>` interface. Understand the responsibilities of each method required by the interface. Sketch a test plan for a class that implements this interface. The class you will be testing is `Heap12`, which implements `PQueue`, and which you will be defining as part of this assignment (see the next section). So, your plan should involve testing all the methods required by the `Heap12` class.

Define a class named `Heap12Tester` that extends `junit.framework.TestCase` and that implements your test plan. Keep in mind that while you should definitely think about testing first, you do not need to write a complete tester program before starting to define `Heap12`. In fact, an iterative process can work well: define some tests in `Heap12Tester`, and implement the functionality in `Heap12` that will be tested by those tests, test that functionality with `Heap12Tester`, write more tests, etc. The end result will (hopefully!) be the same: A good tester, and a good implementation of the class being tested.

Also keep in mind that your Heap12Tester must not make use of any Heap12 methods not specified in the

PQueue and Heap12 documentation.

Heap12.java

Define a generic class `Heap12<E extends Comparable<? super E>>` that implements the `PQueue<E>` interface. Besides the requirements for the methods documented in that interface, and in the Javadoc documentation for `Heap12<E extends Comparable<? super E>>`, your `Heap12` must meet these requirements:

1. `Heap12` must use a heap-structured array as its backing store.
2. `Heap12` must define a public default constructor which creates an array of length 5 as its initial backing store.
3. `Heap12` must define a copy constructor, which takes another Heap12 as arugment, and which initializes the new Heap12 to be a deep copy of the argument Heap12. Here "deep copy" means that the new Heap12 and the argument Heap12 will have different backing store arrays, but the data objects pointed to by elements of those arrays will not be copied.
4. When the `add()` method is called with a non-null argument and the current size of the `Heap12` is equal to the length of its backing store array, the length of its backing store array must double. (That is, a new array with length twice that of the existing full array must be allocated, elements from the old array copied into it, and the new array then used as the backing store.) Thus a `Heap12` object appears to a client as a priority queue with unlimited capacity.

Think about what other methods it might be useful to define in order to implement the required methods. Since these other methods are not part of the public interface of your `Heap12` class, they should have private visibility. For example, methods

```
private void trickleDown(int indx)
private void bubbleUp(int indx)
```

that can perform the 'trickledown' and 'bubbleup' heap operations on an element in the array backing store with a given index might be useful. These methods can be defined recursively, or iteratively.

In addition, it might be useful to define a private constructor that the static `sort()` method can call to create a Heap12 object initialized with a particular given array. Consider that the `sort()` method is very easy to define in terms of `Heap12` public methods, but it has to sort its argument array without allocating another array, while the default public `Heap12` constructor does allocate another array (of length 5).

An Application of Priority Queues

Priority queues are used as supporting data structures in many advanced algorithms. One example is heapsort, which you are implementing as part of this assignment. Another is the Huffman coding algorithm, which constructs a code for representing symbols from a given information source. The code constructed by the Huffman algorithm is optimal, in that it uses the fewest number of bits (i.e. the smallest space) possible (subject to some assumptions). As a result, Huffman coding is useful as part of the JPEG image compression and the mp3 audio compression standards, in the Unix file compression command `pack`, and other applications.

In the `public/P4` directory are two .jar file implementations of Huffman coding for file compression. (These

programs don't actually compress a file, but they construct the Huffman code for the file and show how much it would be compressed.) To run them, copy them to your P4 working directory, and:

```
java -jar HuffCompress.jar somefile
```

```
java -jar HuffCompress12.jar somefile
```

where `somefile` is the name of a file. The second one, `HuffCompress12.jar`, uses your `Heap12.class` in the current working directory as a priority queue in its algorithm; `HuffCompress.jar` uses its own, built-in priority queue. If your `Heap12` is working correctly, they should give the same results!

Turning in your assignment

After you are done with the assignment, you can turn it in. The process is essentially the same as for previous assignments; when logged into ieng6 or a lab workstation, cd to the directory containing your solution file and run the command

```
bundleP4
```

and carefully follow the dialog.

Grading

There are 15 possible points on this assignment, not including early-turnin points. We will test your solution as follows:

- We will copy your turned-in files into an otherwise empty directory.
- Then we will copy in the `PQueue.java` file from the `public/P4` directory.
- We will then compile your solution with the commands:

  ```
  javac Heap12Tester.java
  javac Heap12.java
  ```

  (If your solution files do not compile, you will recieve 0 points for this assignment.)
- We will test your `Heap12` class to make sure that it implements correctly the requirements of this assignment.
- We will test your `Heap12Tester` program to make sure it thoroughly tests a `Heap12` class's implementation of `PQueue`.

We will also consider your coding style, including commenting. Commenting and style guidelines are the same as for previous assignments. Keep in mind that the javadoc comment on a public class should use the `@author` tag to indicate the name of the author of the code (that is, you).

Good luck!