About Lab 8

In Lab 8 we implement and apply priority queues. The class you build is MyPriorityQueue<T>, which is implemented om top of an ArrayList<T>.

The constructor for this is

MyPRiorityQueue(int initial_capacity, Comparator<T> cmp);

Use the first argument as the initial capacity for an ArrayList<T> that holds the queue's data. The constructor should save the second argument in a class variable. Data comparisons in the two percolate methods use cmp.compare(T x, T y) rather than the usual "less than" or "greater than" operations.

The three primary methods of priority queues are

- T peek( ), which returns the smallest value in the queue without channging the queue. For this lab we will have peek( ) return null rather than throw an exception if the queue is empty.
- T poll( ) removes the smallest value from the queue and returns it.
- boolean offer(T x) adds element x and then returns true. If we were implementing priority queues in arrays instead of ArrayLists offer(x) would return false if there was no room in the array for x.

As we discussed in class, you will write private methods percolateUp(int index) and percolateDown(int index) to help with the implementation of poll( ) and offer( )

As we discussed in class, you will write private methods percolateUp(int index) and percolateDown(int index) to help with the implementation of poll( ) and offer( )

To test your priority queue you will need a comparator. Here is a very simple one:

```
class SimpleComp implements Comparator<Integer> {
        int compare( Integer x, Integer y) {
                if (x < y)
                        return -1;
                else if (x == y)
                        return 0;
                else
                        return 1;
        }
}
```

You can test your queue implementation by creating a queue:

```
MyPriorityQueue<Integer> pq = new
        MyPriorityQueue<Integer>(20, new SimpleComp() );
```

Giving it some data:

```
pq.offer(25);
pq.offer(7);
```

etc.

and then polling it in a loop:

```
while (pq.size() > 0)
        System.out.printf(pq.poll() );
```

This should print the data you offered from smallest to largest.

The last part of lab 8 shows how priority queues might be used by an operating system to implement scheduling algorithms. You are given a Task class with a number of items:

String name; (the task's name, such as "Give Bob Money")
int priority; (larger values have higher priority)
int availableTime; (when the task can be started)
int length; (how long the task takes)
int deadline; (when the task should be completed)

You are given an AvailableTime comparator; a priority queue based on that will list the tasks in order of when they can be started/

Part 3 of the lab asks you to implement four more Task comparators. Each of these goes in its own file.

There is a Scheduler program that reads a file of tasks, schedules them according to one of the comparators, and then prints some information about how effective the comparator was as a scheduler.

This program is complete; you don't need to make changes to it. The arguments to it that you will need to set in the RunConfiguration are

a) The name of the task file (one of jobs10.txt, jobs100.txt, and jobs1000.txt)
b) A string that implements which comparator to use (One of "priority" "available", "deadline", "length", "name").