COMPUTER SCIENCE 21A (FALL TERM, 2015)
DATA STRUCTURES

PROGRAMMING ASSIGNMENT 3

Due Saturday, Dec 12th at 11pm.

VERY IMPORTANT

- Your code should be very well commented:
  - Add your name and email address at the beginning of each .java file.
  - Write comments within your code when needed.
    - You must use Javadoc comments for your classes and methods.
  - In each method's Javadoc header, include its running time.
- Before submitting your assignment you **must** zip all your files.
  - The ZIP file must have the naming convention LastnameFirstname-PA2.zip
- Submit your assignment on LATTE using the assignment submission link.
- You may only use data structures and algorithms that you implement yourself.
- Your structures **do not** need to support generic types.

Part 1: Minimum Priority Queue

Implement a `MinPriorityQueue` class that will hold some number of Processes.
The operations a `MinPriorityQueue` should have are:

1. The ability to construct a minimum priority queue of size *n*.
2. The ability to enqueue an ID based on its priority.
3. The ability to dequeue the ID associated with the minimum priority.

Remember that a priority queue is implemented by using a heap, so you will have to include other operations that will help you with the enqueue and dequeue operations. The priority queue should NOT allow for duplicates. In order to support this requirement, you will be using a hash table to check for duplicates.

Just as a convention: when doing heapify-down, if the two children have the same priority, and you need to swap with one of them, swap with the left child.

A priority of 0 indicates the highest priority and a priority of infinity indicates the lowest priority.

Part 2: Hash Table

Implement a `HashTable` class. Your hash table should store key-value pairs with a process ID as the key and a process object as the value. The collisions are handled by open addressing with cuckoo hashing.

The operations a `HashTable` should have are:

1. The ability to construct a hash table initialized with size *n*.
2. The ability to create a key based on a hash function.
3. The ability to add a record containing a key and a value at the position indicated by the result of the hash function.
4. The ability to search for a value.
5. The ability to delete the key-value pair from the hash table.

Remember to rehash once the hash table gets too full. You should maintain a load factor of 60%.

Task: Print Processes based on priority

Your task is to:

1. Prompt the user for an input file.
2. In the input file, each line contains (in this order), an ID, a priority value, and a name for the Process. Store each line as a Process object.
3. Add each Process to a hash table of initial size 2029 where the key is the ID of a Process and the value is the Process object. If this matches an existing entry, abort the modification of the hash table and skip enqueueing the Process in the next step.
4. If the Process handled in step 3 was not a duplicate, enqueue its ID in a min priority queue of size 62887.
5. Dequeue each ID from the min priority queue. When you dequeue an ID, retrieve its respective Process object from the hash table and then possibly add it to the output according to step 6.
    1. After dequeuing, you must delete the Process object from the hash table. Be careful with the delete method; it is easy to overlook some cases.
6. Your output should be the names of the last 21 Processes in reverse order of dequeuing from your min priority queue, each separated by a space. Please structure the output by building a String from right to left and then printing out that string when it is completed.
    a. For example, given Processes p1, p2, and p3,
    where p1.priority < p2.priority < p3.priority
    the output would be:
    p3.name + " " + p2.name + " " + p1.name