COMPUTER SCIENCE 21A (SUMMER TERM, 2017)
DATA STRUCTURES AND ALGORITHMS

PROGRAMMING ASSIGNMENT 3

Due Mon, Aug 7

REMEMBER

- Your code should be well commented:
    - Add your name and email address at the beginning of each .java file.
    - Write a clear description of each class.
    - Write comments within your code when needed.
- Before submitting your assignment, zip all your files.
- To submit, upload your zip file on Latte.

Part 1: Load Process information from file

Read the file, `Process.txt`, where each line contains (in this order) an ID, a priority value and a name of the process. Implement a `Process` class that will hold each of these values. The operations a Process should have are:

1. The ability to **construct** a Process.
2. The ability to **return** all values.

Part 2: Minimum Priority Queue

Implement a `MinPriorityQueue` class. The operations a MinPriorityQueue should have are:

1. The ability to **construct** a min priority queue.
2. The ability to **enqueue** a process ID with its priority.
3. The ability to **dequeue** a process ID.

Remember that a priority queue is implemented by using a heap, so you will have to include other operations that will help you with 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.

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.

Priority 0 indicates the highest priority and priority 20000 indicates the lowest.

Part 3: 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 double hashing. The operations a hash should have are:

1. The ability to **construct** a hash table. Initialize it with size 2029.
2. The ability to **create** a key based on a hash function.
3. The ability to **add** a record containing key and 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.

Play around with several different hash functions, and explain why you choose the ones you use.

Remember to rehash once the hash table gets too full!

Task: Print Processes based on priority

Your task is to:

1. Load the `Process.txt` file.
2. Store each process as a Process object.
3. Add each process to the hash table 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 wasn't a duplicate, enqueue it in the min priority queue through storing the ID of the process with the priority of the process.
5. Dequeue each process from the min priority queue. When you dequeue a process, retrieve the Process object from the hash table and then print its name. (While debugging it might be helpful to print its priority too.)
6. After dequeueing you must delete the Process object from the hash table. Be careful with the delete method; it is easy to overlook some cases.
7. Your output should be just the names of each process in order of dequeueing from your min priority queue.