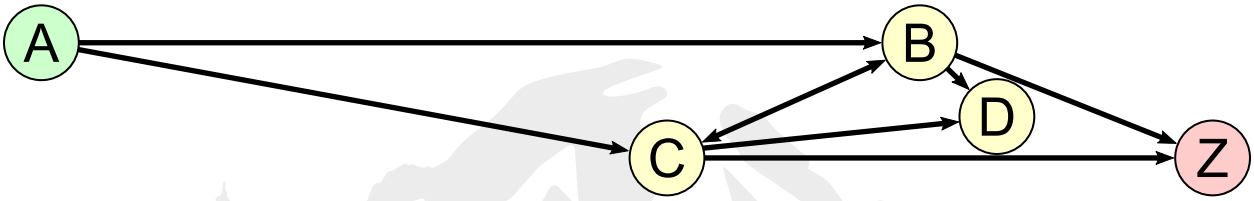
HW 2: Priority Queue for Best-First Search

I have created a best-first-search program, but it is missing the priority queue that it uses to do the search. Your job is to write the missing priority queue class.

The program searches through a graph, such as this one:



The file describing this graph is the single command-line argument for the program. It looks like this:

```
A
A    12   B,C
B    3    C,D,Z
C    6    B,D,Z
D    2
Z    0
```

The first line is the start node. After that, each line contains 3 tab-separated tokens: a unique name, a score telling how far it is from a goal state, and a comma-separated list of other nodes it is attached to. A node with a score of 0 is a goal node. Best-first search is an alternative to breadth-first or depth-first. It searches the nodes in order of their score. The priority queue is used in order to efficiently calculate which node to explore next.

The above graph yields the following output:

```
Loading file "example.graph".
Exploring node A... adding nodes B,C
Exploring node B... adding nodes C,D,Z
Exploring node Z... GOAL!
Upon termination, there were 3 nodes ready to explore:  [(D: 2), (C: 6),
(C: 6)]
```

If you are using Java, your priority queue must implement generics for the type to be added. If you are using a weakly-typed language like Python, you may rely on its looser conventions. If you are using another language, you must recreate the downloadable file in the language of your choice, and turn that in too. Follow typing conventions that make sense for that language.

Your class must contain the following public methods, with the time complexities as shown:

- `PriorityQueue(boolean highFirst)` (constructor) (O(1)). If `highFirst` is `true`,

elements with high scores will be removed first. If it is false, elements with low scores will be removed first.

- `PriorityQueue()` (constructor) ($O(1)$). This will default to making a queue that priorites low scores.
- `void add(E object, int score)` ($O(\log n)$). This adds `object` to the priority queue with its corresponding integer `score`.
- `void clear()` ($O(1)$). This clears out the priority queue, setting its size to 0.
- `boolean contains(E object)` ($O(n)$). This searches the priority queue for the object in question, without changing the structure. It returns `true` if it was found, or `false` if not.
- `int size()` ($O(1)$). This returns the number of elements in the priority queue.
- `E getNext()` ($O(\log n)$). This removes the next item from the priority queue, returning it. It should return `null` if the priority queue is empty.
- `boolean isEmpty()` ($O(1)$), which returns `true` if and only if the size is zero.
- `E peek()` ($O(1)$). This returns the next time from the queue, without changing it. It should return `null` if the priority queue is empty.
- `boolean remove(E object)` ($O(n)$). This searches the priority queue for the object in question, and removes it. It returns `true` if it was successful, or `false` if not. If there are multiple copies of this object, it will only remove one of them.
- `String toString()` ($O(n)$). This returns a string of comma-separated values, representing every item in the priority queue and its score. There is no need for the string to be sorted.

Note that this list uses Java-based syntax. You are responsible for adjusting the functions as needed if you are using a different language. Also, not every method is used by the best-first-search program. However you will be tested on all of them.

All of these methods should do the work needed to maintain the heap property inside the priority queue. You should not have any other public methods, but feel free to make as many supplementary private methods as you want.

The data structure should be in its own file. You may create supplementary classes if you wish. These should be hidden away inside the primary class. The final file that you turn in should be called `PriorityQueue.java`, `priorityqueue.py`, or whatever equivalent is appropriate in your language.

Finally, style matters! Please make your code readable and easy to maintain. Follow all the standards of your language of choice.