CS2400 Data Structures and Advanced Programming
Project 2:
(100 points)

Project requirements:

- Implement the MaxHeapInterface (provided)
    - The name of the class that implements MaxHeapInterface must be, “ArrayMaxHeap”
    - ArrayMaxHeap must implement MaxHeapInterface using an array
    - You must implement a private method called, “maxHeapify”, to adjust the heap in the case of a removal. This method must conform to the algorithm provided.
    - You must implement a private method called “maxUpHeap”, to adjust the heap in the case of an addition. This method must conform to the algorithm provided.
    - Tus data structure must dynamically resize.

- Implement the PriorityQueueInterface (provided)
    - The class that implements PriorityQueueInterface must be called, “HeapPriorityQueue”.
    - The storage structure for this implementation is the above mentioned ArrayMaxHeap.
    - There must be a private property of type “MaxHeapInterface” that will reference an instance of the ArrayMaxHeap object. This will be where all the objects are stored in the priority queue.

- **Extra Credit (5 points):** create a constructor (in addition to a default constructor) for both ArrayMaxHeap and HeapPriorityQueue that takes an array of type T as an argument, copy its elements to the internal storage array and use a method called, “buildMaxHeap”, to build it into a heap. The buildMaxHeap method must conform to the provided algorithm. Do not use the add method to accomplish this. No pattial extra credit will be awarded. If you choose to do this part you must include tests for it to get credit.

- **Be sure not to alter the interface files in any way**

Testing:

- Write a driver class with a “main” method that thoroughly tests your implementation of the MaxHeapInterface. Call this class, “MaxHeapInterfaceTest”.
- Write a driver class with a “main” method that thoroughly tests your implementation of the PriorityQueueInterface. Call this class, “PriorityQueueInterfaceTest”.
- In these classes be sure to write each test case in a separate appropriately named method. Each test method should be called from the main method. Be sure that when you define a reference variable for each data structure, the variable is of the interface type.
- You can create a single instance of the object under test in the main method and pass it as an argument to the test method calls or you can create a new instance of the object in each method that is called, or you can create a class variable and instantiate it in main.
- If you chose to do the extra credit constructor you must include tests for it to get credit.
- Be sure to test your code thoroughly; this project will be graded more strictly than previous assignments.
- Be sure that your code uses the compareTo method and not comparison operators to compare stored objects.

Report:

- Write a report that summarizes your work on this project.
    - Write about the difficulties that you encountered while implementing the various parts of this project.
    - Write about your approach to testing the implementations. Explain the edge cases that you had to test and the methods that you used to debug the implementations.
    - If you chose to implement the extra credit part then discuss how you chose to test it.
    - This report should be at least three pages not including the cover and introduction pages.
    - Take this report seriously as it will be a larger portion of your project grade than it was in previous assignments.
- Make sure that you include a cover page and an introduction page.

What to submit:

- Your report (in PDF format)
- Compress all of the.java files and the report into a single file called YourFullName_p2.zip
- Only submit the zip file via Blackboard; do not submit other files.
- **Note:**
    - **Late submissions will not be accepted for any reason. Start this project right away.**
    - **Code that does not compile will not be accepted.**

Algorithms:

```
maxHeapify(array, index){
    if index > heapsize / 2
        return
    endif

    largest = index
    leftChildIndex = 2 * index
    rightChildIndex = (2 * index) + 1

    if leftChildIndex <= heapSize AND array[leftChildIndex] > array[largest]
        largest = leftChildIndex
    end if

    if rightChildIndex <= heapSize AND array[rightChildIndex] > array[largest]
        largest = rightChildIndex
    end if

    if largest != index
        swap(array[index], array[largest])
        maxHeapify(array, largest)
    end if
end maxHeapify

maxUpHeap(array, index)
    while index > 1 AND array[index/2] < array[index]
        swap(array[index/2], array[index])
          index = index / 2
     end while
end maxUpHeap

buildMaxHeap(array)
    size = number of items stored
    for index from (size / 2) to 1
        maxHeapify(array, index)
    end for
end buildMaxHeap
```