Data Structures	**Programming Project 3**
CSC2-242 Fall 2025 (Ali Azhari)	**Due Date: Sunday Nov. 9, 2025,**

Please solve the problem(s) alone. Remember to test your solution thoroughly. Code that does not work correctly will lose credit. Code that does not compile will receive no more than 70%. Please submit a digital copy, of your source code (Moodle). The digital copy of your code should be a zip file of the project folder named with your last name followed by the project number. For example, azhari3.zip would be my zip file for project 3. Good luck!

I introduce a new abstract data type known as a priority queue.
This is a collection of prioritized elements that allows arbitrary element insertion and allows the removal of the element that has first priority. When an element is added to a priority queue, the user designates its priority by providing an associated key. The element with the minimal key will be the next to be removed from the queue (thus, an element with key 1 will be given priority over an element with key 2).

There are many strategies to implement PQ to make it more efficient and to satisfy our needs. As we discussed in class, the implementation of PQ using a data structure called Binary Heap is more efficient.

However, there are situations in which additional methods would be useful, as shown by the scenarios below involving the standby airline passenger application.

- A standby passenger with a pessimistic attitude may become tired of waiting and decide to leave ahead of the boarding time, requesting to be removed from the waiting list. Thus, we would like to remove from the priority queue the entry associated with this passenger. Operation removeMin does not suffice since the passenger leaving does not necessarily have first priority. Instead, we want a new operation, remove, that removes an arbitrary entry.
- Another standby passenger finds her gold frequent-flyer card and shows it to the agent. Thus, her priority has to be modified accordingly. To achieve this change of priority, we would like to have a new operation replaceKey allowing us to replace the key of an existing entry with a new key.
- Finally, a third standby passenger notices her name is misspelled on the ticket and asks it to be corrected. To perform this change, we need to update the passenger's record. Hence, we would like to have a new operation replace-Value, allowing us to replace the value of an existing entry with a new value.

Phase I: upheap – downHeap

You are provided with a complete ADT implementing PQ using ADT Binary Heap. You will only have to implement:

- upHeap(int j)
- downHeap(int j)
- parent(int j)
- left(int j)
- right(int j)
- hasLeft(int j)
- hasRight(int j)

Phase II. Adaptable Heap

The above scenarios motivate the definition of a new adaptable priority queue ADT that extends the priority queue ADT with additional functionality.

In order to implement methods remove, replaceKey, and replaceValue efficiently, we need a mechanism for finding a user's element within a priority queue, ideally in a way that avoids performing a linear search through the entire collection.

In the original definition of the priority queue ADT, a call to insert(k, v) formally returns an instance of type Entry to the user. In order to be able to update or remove an entry in our new adaptable priority queue ADT, the user must retain that Entry object as a token that can be sent back as a parameter to identify the relevant entry. Formally, the adaptable priority queue ADT includes the following methods (in addition to those of the standard priority queue):

- remove(e): Removes entry e from the priority queue. If the entry is the last element of the ArrayList then simply removes the element from the ArrayList by calling the method remove that belongs to the ArrayList. If the entry is in the middle or top of the queue, then you want to switch it with the last element. Now the last element takes the place of the entry removed. Be careful, the current entry now may not be in its right position of the queue. You can either upheap or downHeap. **Use the bubble method That you also need to implement.**
- replaceKey(e, k): Replaces the key of existing entry e with k. This method also might use bubble to position the entry with the replaced key in its rightful position.

- replaceValue(e, v): Replaces the value of existing entry e with v. All you need to do is locate the entry by calling the method locate and then change the value of that location.

An error occurs with each of these methods if parameter e is invalid (for example, because it had previously been removed from the priority queue).

General Reminders

If you desire, you may add private methods to help you with writing the code. You may not however add or subtract from any public interface nor are you allowed to remove private methods.

- Start early
- Download the zip file named project3.zip.
- Project3.zip contains the following files:
  - ☐ Entry.java. (Interface)
  - ☐ PriorityQueue.java (Interface)
  - ☐ AbstractPriorityQueue.java (Abstract class)
  - ☐ HeapPriorityQueue.java (Concrete class) (to be modified)
  - ☐ AdaptablePriorityQueue.java (Interface)
  - ☐ HeapAdaptablePriorityQueue.java (Concrete class) (To be modified)

- Make sure to check your code against the rubric before submission. ▪
- You are encouraged to see me for help with the project.

Submission

You are provided with the following interfaces/Abstract/Concrete classes.
You are not allowed to change any of the codes except those that are required to be modified (TODO)

Task:

- HeapPriorityQueue.java: Implement upHeap method.
- HeapPriorityQueue.java: Implement downHeap method.
- HeapPriorityQueue.java: Implement left(int j), right(int j), parent(int j), hasLeft(int j), and hasRight(int j);
- HeapAdaptablePriorityQueue.java: Implement bubble method
- HeapAdaptablePriorityQueue.java: Implement remove(entry) method.
- HeapAdaptablePriorityQueue.java: Implement replaceKey(entry, key).
- HeapAdaptablePriorityQueue.java: Implement replaceValue(entry, value).

Digital Submission: Since our projects now have multiple files that need to be graded, I ask that you submit a zip file containing Only the HeapPriorityQueue.java and HeapAdaptablePriorityQueue.java. Failure to follow these directions will result in a deduction of up to 30%.

Grading

- upHeap (20 points)
- downHeap (20points)
- bubble (10 points)
- left, right, parent, hasLeft, and hasRight (10 points)
- remove(entry) (10 points)
- replaceKey(entry, key) (10 points)
- replaceValue(entry, value) (10)
- Program passes tests (10 points)

Good luck!