Due Nov 12 by 11:59pm **Points** 100 **Available** after Oct 29 at 1pm

Homework 6: Recursive Data Structures: Lists

Home

Modules

Homework 6: PriorityQueue w/ Recursive Data Structure (List)

PriorityQueue

1 Introduction

Your task is to implement an immutable **`Priority Queue`** (PQ). A priority queue is a data structure, where every element of a PQ contains two properties:

1. A priority - an **`Integer`**
2. A value associated with the priority - for this assignment, the value will be a **`String`**.

The PQ interface is included below. You must implement this protocol using a **concrete class** called **`ListPriorityQueue`**. Your implementation must use your own **linked-list recursive data structure** (similar to what we developed in lecture this week). You are free to use the lecture code shared by Prof. Keith as a starting point for this assignment. You **may NOT use any of Java's built-in collections (e.g., LinkedList) or maps (e.g., HashMap)**.

2 What to do

Your PriorityQueue (PQ) implementation must support the following ADT operations (note that we're using Java's "boxed" - **`Boolean`** and **`Integer`** - instead of Java's primitive types for the method signatures):

- **`PriorityQueue createEmpty()`**: Creates and returns an empty PQ.
  Special Note: the createEmpty method should be a public class method (static) in your concrete class. It is NOT part of the PQ interface.
  In a future lecture, you'll learn that we call these "factory methods" but for now ensure you supply a static method in your concrete class that returns an empty PQ.
- **`Boolean isEmpty():`** Checks if the PQ is empty. Returns **true** if the PQ contains no items, **false** otherwise.
- **`PriorityQueue add(Integer priority, String value):`** Adds the given element (the priority and its associated value) to the PQ. The range of acceptable values for our PQ is 1..10. Any values outside of this range should throw an **`IllegalArgumentException`**.
  Our in-lecture List implementation utilized an insertion sort; you might want to consider something similar here. Your PQ should be ordered from the highest to lowest priority.

- `String peek()`: Returns the value in the PQ that has the highest priority.
  - For two positive integers, i and j. If i < j then i has a lower priority than j.
  - After a call to `peek()` the PQ remains unchanged.
  - Calling `peek()` on an empty PQ should throw an `EmptyPriorityQueueException`.
- `PriorityQueue pop()`: Returns a copy of the PQ without the element with the highest priority.
  - Calling `pop()` on an empty PQ should throw an `EmptyPriorityQueueException`.

Multiple elements in the PQ may have the same priority, which will impact `peek()` and `pop()`. If there are multiple elements that have the same priority, return the value of the earliest added element.

Similar to the recursive data structure we worked on in lecture, your PQ should be immutable. Also, although we have not explicitly specified overrides for `toString()`, `equals()` and `hashCode()`, you might want to consider those - particularly if your code does not pass certain tests on the Gradescope server.

Your solution code must be placed in a package named `cs5004.collections` (note the two levels of packages for this assignment)

```
package cs5004.collections;

public interface PriorityQueue {


  /**
   * Checks if the priority queue is empty
   * @return true if the PQ is empty, false otherwise.
   */
  Boolean isEmpty();

  /**
   * Adds an element to the PQ.
   * @param priority The element's (non-negative) priority.
   * @param value The element's value.
   * @return A copy of the priority queue containing the new element.
   */
  PriorityQueue add(Integer priority, String value) throws IllegalArgumentException;

  /**
   * Gets the value of the highest priority element. If there are multiple elements that have the same priority, gets
   * the value  of the most recently added element.
   * @return The value  of the highest priority element.
   * @throws EmptyPriorityQueueException if the PQ is empty.
   */
```

```
  String peek() throws EmptyPriorityQueueException;

  /**
   * Removes the highest priority element.
   * @return A copy of the priority queue without the highest priority element.
   * @throws EmptyPriorityQueueException if the PQ is empty.
   */
  PriorityQueue pop() throws EmptyPriorityQueueException;
}
```

We also have a custom Exception for this assignment called **`EmptyPriorityQueueException`**. Include this exception with your code to match our specification

```
package cs5004.collections;

public class EmptyPriorityQueueException extends Exception {
}
```

3 Important Notes

Be sure to include a UML class diagram that illustrates the structural design of your solution.
Again, you are **NOT** allowed to use any of the Java collections you used in previous assignments and your solution must be a recursive data structure. This exercise it intended to give you experience "building stuff under the hood". Use of the Java collections will result in hefty deductions from your overall score.
You have liberty with respect to your concrete implementation as long as you

- name the concrete class as indicated,
- adhere to the PQ protocol (interface) given,
- provide us with the public, class-based factory method as indicated.

Our auto-tests will be operating based on the protocol. We will manually assess your overall design and unit tests. Ensure your unit tests are **NOT** created in the cs5004.collections package. Your tests should have the same access as any external client.

4 What to submit

1. You can upload your homework to Gradescope by create a zip file that contains the src and test directories (with the appropriate package directories as specified in this assignment)

   OR

2. You can submit to Gradescope via your GitHub account if you have one. As part of your academic honesty agreement, ensure your GitHub repo is **private**, not public!