Assignment 6

Refer to Canvas for assignment due dates for your section.

Objectives:

- Write recursive implementations of ADTs.
- Test ADTs.

General Requirements

Create a new Gradle project for this assignment in your course GitHub repo. Make sure to follow the instructions provided in “Using Gradle with IntelliJ” on Canvas.

Create a separate package for each problem in the assignment. Create all your files in the appropriate package.

To submit your work, push it to GitHub and create a release. Refer to the instructions on Canvas.

Your repository should contain:

- One .java file per Java class.
- One .java file per Java test class.
- UML diagrams for each problem. UML diagrams can be generated using IntelliJ or hand-drawn.
- All non-test classes and non-test methods must have valid Javadoc.

Your repository should **not** contain:

- Any .class files.
- Any .html files.
- Any IntelliJ specific files.

For both problems in this assignment, your underlying data structure must be recursive. You may not use any of Java’s built-in collections (e.g., LinkedList) or maps (e.g., HashMap).

Problem 1

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 - in our case the value will be a String.

Your PQ implementation must support the following ADT operations:

- `PriorityQueue createEmpty()`: Creates and returns an empty PQ.
- `Boolean isEmpty()`: Checks if 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.
- `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`. The PQ remains unchanged. Calling `peek()` on an empty PQ should throw an exception.
- `PriorityQueue pop()`: Returns a copy of the PQ without the element with the *highest* priority. Calling `pop()` on an empty PQ should throw an exception.

Multiple elements in the PQ may have the same priority, which will impact `peek()` and `pop()`. You may choose how to handle this situation but be sure to describe how you handle it in the documentation for the affected methods.

Problem 2

You have been hired to help a start-up Natural Language Processing (NLP) team develop fundamental code for their Bag-of-Words Model. In the bag-of-words model, some text is represented as a multiset (a bag) of its words, where we disregard grammar and often also the order of words.

Your job is to design and implement the first version of an immutable `BagOfWords`. A `BagOfWords` is a data container, holding `Strings` (words). A `BagOfWords` can contain duplicates, and `Strings` (words) have no order.

Here is the detailed specification of the `BagOfWords` ADT.

- `BagOfWords emptyBagOfWords()`: Creates and returns an empty `BagOfWords`.
- `Boolean isEmpty()`: Checks if the `BagOfWords` is empty. Returns true if the `BagOfWords` contains no items, false otherwise.
- `Integer size()`: Returns the number of elements in the `BagOfWords`. Duplicates should be counted as separate elements e.g. if the `BagOfWords` contains “frog”, “frog”, “toad”, the size would be 3.
- `BagOfWords add(String s)`: Returns a new `BagOfWords` that contains all elements in the original `BagOfWords` plus `s`.
- `Boolean contains(String s)`: Checks if `s` is in the `BagOfWords`. Returns true if the `BagOfWords` contains `s` and false otherwise.

As always, your implementation should include the `equals()` method. To determine if two `BagOfWords` are equal, remember that the order of words stored in `BagOfWords` does not matter. If the exact same elements are present in both `BagOfWords`, they should be equal.