Homework Assignment #8

CS5004 – Object-Oriented Design
Northeastern University – Silicon Valley
Summer 2020

Due Sunday 07/19 at 11:00pm PT

Grading: Each programming problem is graded as follows

- A submission which does not compile gets 0.
- A submission which compiles but does something completely irrelevant gets 0.
- A submission which works (partially) correctly, gets (up to) %80 of the total credit.
- %20 is reserved for the coding style. Follow the coding style described in the book.

Problem 1 [40pts]. In the class, we saw the generic class `Pair`. In this problem, the objective is to implement class `Triple` which holds three heterogeneous objects (i.e. they are not necessarily of the same type). Implement all the usual methods. Note that you do not need to implement `compareTo()`.
Submission format: You must submit two files: `Triple.java` which contains the `Triple` class and `TripleTest.java` which contains the class `TripleTest` whose `main()` method should test the functionality of `Triple`. Create a few tests for each method.

Problem 2 [90pts]. Implement a priority queue capable of holding objects of an arbitrary type `T` by defining a generic `PriorityQueue` class. A priority queue is a type of container where every item also has an associated weight or priority. For the scope of this problem, the type of priority value can be anything as long as its values can be compared (`Comparable`). Note that larger values mean higher priorities. Your class should support the following methods:

- `add(item, priority)` Adds a new item to the queue with the associated priority.
- `peek()` Returns the item with the highest priority. Returns `null` if empty.
- `remove()` Returns and removes the highest-priority item. Returns `null` if empty.
- `reset()` Empties the priority queue removing all elements.

For example, if `q` is a priority queue defined to take `String`s

```
q.add("X", 10);
q.add("Y", 1);
q.add("Z", 3);
System.out.println(q.remove()); // Returns X
System.out.println(q.remove()); // Returns Z
System.out.println(q.remove()); // Returns Y
```

Additionally, your class must support

- `size()` Returns the number of elements.
- `toString()` Returns a string representing all elements in the priority queue.
- `equals(obj)` Returns `true` if `obj` is another priority queue with the *same* elements.

Obviously, you must provide a default constructor. Moreover, you need to write a copy constructor as well which makes an identical (deep) copy of its argument.

Note 1: Use an `ArrayList` to store the elements internally. Try thinking of an efficient way to keep track of priorities. There are many ways to implement this. Remember that when the highest-priority element is removed, you need to "know" or find the next highest-priority element.
Note 2: Priority queue is an unordered data structure. Therefore, even though you can use an `ArrayList`, the `equals()` method should work as expected. Notice that, we can store duplicates in the `PriorityQueue`. With regard to the `equals()` method, this has to be taken into account. For instance, if one `PriorityQueue` has two 3's, the other one must have two 3's as well.
Note 3: If there are two or more elements with the same priority, any one of them can be returned. Make sure that `peek()` and `remove()` return the same element in all cases.
Submission format: Create a package called `utility` which contains `PriorityQueue.java`. This package is to be imported by `PQTest.java` shown below. Use `java -ea` to test. We will check your code with other tests too!

```
import utility.PriorityQueue;

public class PQTest {
    public static void main(String[] args) {
        try {
            PriorityQueue<String, Integer> p1 = new PriorityQueue<>();
            p1.add("coffee", 10);
            p1.add("milk", 8);
            p1.add("tea", 11);
            p1.add("cake", 22);
            p1.add("sake", 3);

            assert(p1.size() == 5);
            assert(p1.peek() == "cake");
            p1.remove();
            assert(p1.peek() == "tea");
            p1.add("matcha", 11);
            String s = p1.remove();
            assert(s.equals("tea") || s.equals("matcha") == true);

            p1.reset();
            assert(p1.size() == 0);
            assert(p1.remove() == null);
            PriorityQueue<String, Integer> p2 = new PriorityQueue<>();

            for(int i = 0; i < 10; i++) {
                Character x = (char) ('a' + i);
                Character y = (char) ('a' + 9 - i);
                p1.add(x.toString(), i);
                p1.add(x.toString(), i);
                p2.add(y.toString(), 9 - i);
                p2.add(y.toString(), 9 - i);
            }

            assert(p1.size() == 20);
            assert(p2.size() == 20);
            assert(p1.equals(p2) && p2.equals(p1));

            p1.remove();
            assert(!(p1.equals(p2) || p2.equals(p1)));

            // Here the type of the priority values is String
            PriorityQueue<Integer, String> p3 = new PriorityQueue<>();
            p3.add(1, "a");
            p3.add(2, "A");
            assert(p3.peek().equals(1));
        } catch(AssertionError e) {
            System.err.println("Does not pass.");
            System.exit(1);
        }

        System.out.println("Passes.");
    }
}
```