CSE 143: Computer Programming II

QuickCheck: Collections Solutions (due Thursday, February 5)

0. Implementing `union`

Write a method `union` that accepts two maps (whose keys and values are both integers) as parameters, and returns a new map that represents a merged union of the two original maps. For example, if two maps `m1` and `m2` contain these pairs:

```
m1:  {7=1, 18=5, 42=3, 76=10, 98=2, 234=50}
m2:  {7=2, 11=9, 42=-12, 98=4, 234=0, 9999=3}
```

The call of `union(m1, m2)` should return a map that contains the following pairs:

```
{7=3, 11=9, 18=5, 42=-9, 76=10, 98=6, 234=50, 999=3}
```

The "union" of two maps `m1` and `m2` is a new map that contains every key from `m1` and every key from `m2`. Each value stored in your "union" map should be the sum of the corresponding value(s) for that key in `m1` and `m2`, or if the key exists in only one of the two maps, that map's corresponding value should be used. For example, in the maps above, the key 98 exists in both maps, so the result contains the sum of its values from the two maps, $2 + 4 = 6$. The key 9999 exists in only one of the two maps, so its sole value of 3 is stored as its value in the result map.

You may assume that the maps passed are not null, though either map (or both) could be empty. Though the pairs are shown in sorted order by key above, you should not assume that the maps passed to you store their keys in any particular order, and the map you return does not need to store its keys in any particular order.

Solution:

```java
public Map<Integer, Integer> union(Map<Integer, Integer> m1, Map<Integer, Integer> m2) {
   Map<Integer, Integer> result = new HashMap<Integer, Integer>();
   for (Integer key : m1.ketSet()) {
      result.put(key, m1.get(key));
   }
   for (Integer key : m2.keySet()) {
      if (!result.containsKey(key)) {
         result.put(key, m2.get(key));
      } else {
         result.put(key, result.get(key) + m2.get(key));
      }
   }
}
```