Growing Separate Chaining Map

The separate chaining map presented in the earlier chapter is simplified – after adding many pairs, we will soon find that the linked lists in each of the indices of the array grows really long. That degrades the performance of the separate chaining map.

Instead, we should have an array that grows bigger whenever the number of pairs in a map get too large with respect to the array size. This ensures that on average, the linked lists in a separate chaining map is short.

We can do so via an `upsize()` function in the `SeparateChainingMapGrow`. The code for this is rather self-documenting, so I leave it here for you to read on your own.

```
// SeparateChainingMapGrow.java

import java.util.LinkedList;
import java.util.List;

public class SeparateChainingMapGrow<K extends Comparable<? super K>, V>
implements Map<K, V> {

  public static final int SCALE_FACTOR = 2;
  public static final int INITIAL_TABLE_SIZE = 8;
  public static final double MAX_LOAD_FACTOR = 1.0;

  private LinkedList<Pair<K, V>>[] table;

  @SuppressWarnings("unchecked")
  public SeparateChainingMapGrow(int initialSize) {
    table = (LinkedList<Pair<K, V>>[]) new LinkedList[initialSize];
    for (int i = 0; i < table.length; i++) {
      table[i] = new LinkedList<Pair<K, V>>();
    }
  }

  @SuppressWarnings("unchecked")
  public SeparateChainingMapGrow() {
    table = (LinkedList<Pair<K, V>>[]) new LinkedList[INITIAL_TABLE_SIZE];
    for (int i = 0; i < table.length; i++) {
      table[i] = new LinkedList<Pair<K, V>>();
    }
  }

  public void put(K key, V value) {
    int index = getIndex(key, table.length);
```

```
    for (Pair<K, V> pair : table[index]) {
      if (key.equals(pair.key)) {
        pair.value = value;
        return;
      }
    }
    if (getSize() / (double) getTableSize() > MAX_LOAD_FACTOR) {
      upsize();
    }
    table[index].add(new Pair<K, V>(key, value));
  }

  public V get(K key) {
    int index = getIndex(key, table.length);
    for (Pair<K, V> pair : table[index]) {
      if (key.equals(pair.key)) {
        return pair.value;
      }
    }
    return null;
  }

  @SuppressWarnings("unchecked")
  public void upsize() {
    LinkedList<Pair<K, V>>[] newTable = (LinkedList<Pair<K, V>>[]) new
    LinkedList[getTableSize() * SCALE_FACTOR];
    for (int i = 0; i < newTable.length; i++) {
      newTable[i] = new LinkedList<Pair<K, V>>();
    }
    for (LinkedList<Pair<K, V>> list : table) {
      for (Pair<K, V> pair : list) {
        int index = getIndex(pair.key, newTable.length);
        // we don't need to check for overwrites since we know for this existing
        // table, pairs have unique keys
        newTable[index].add(pair);
      }
    }
    table = newTable;
  }
}
```