Prepared by Linan Qiu <lq2137@columbia.edu>

Separate Chaining Map

The separate chaining map is simply an extension of the hashtable shown earlier.

SeparateChainingMap

We have an **array of linked lists**. Hence, each element of the array is an empty linked list of Pairs upon initialization.

```
// SeparateChainingMap.java

import java.util.LinkedList;

public class SeparateChainingMap<K extends Comparable<? super K>, V>
implements Map<K, V> {
  private LinkedList<Pair<K, V>>[] table;

  @SuppressWarnings("unchecked")
  public SeparateChainingMap() {
    table = (LinkedList<Pair<K, V>>[]) new LinkedList[8];
    for (int i = 0; i < table.length; i++) {
      table[i] = new LinkedList<Pair<K, V>>();
    }
  }

  @Override
  public void put(K key, V value) {

  }

  @Override
  public V get(K key) {

  }
}
```

put()

In put(), we are trying to add a new Pair of key and value. We perform the following steps:

- Find `int hash = key.hashCode()`, the `hashCode` of the **key of the `Pair`**. Note that we are not finding the `hashCode` of the entire `Pair`. Instead, we only use the `hashCode` of the key – this is because the position of the `Pair` should be determined solely by the `key`
- Find the `index` of the `Pair` in the array of linked lists. This means that `table[index]` will be the linked list that we should put our `Pair`
- Iterate through the linked list just like we did in `ListMap` and overwrite / add. However, this time, we are iterating over a much smaller linked list.

This results in this `put()` method:

```java
// SeparateChainingMap.java

import java.util.LinkedList;

public class SeparateChainingMap<K extends Comparable<? super K>, V>
implements Map<K, V> {
  private LinkedList<Pair<K, V>>[] table;

  @SuppressWarnings("unchecked")
  public SeparateChainingMap() {
    table = (LinkedList<Pair<K, V>>[]) new LinkedList[8];
    for (int i = 0; i < table.length; i++) {
      table[i] = new LinkedList<Pair<K, V>>();
    }
  }

  @Override
  public void put(K key, V value) {
    int hash = key.hashCode();
    int index = hash % table.length;
    if (index < 0) {
      index += table.length;
    }
    LinkedList<Pair<K, V>> list = table[index];

    for (Pair<K, V> pair : list) {
      if (pair.key.equals(key)) {
        pair.value = value;
        return;
      }
    }
    list.add(new Pair<K, V>(key, value));
  }

  @Override
  public V get(K key) {
  }
}
```

`get()`

`get()` is implemented in a similar manner:

- Find `int hash = key.hashCode()`, the `hashCode` of the **key of the `Pair`**. Note that we are not finding the `hashCode` of the entire `Pair`. Instead, we only use the `hashCode` of the key – this is because the position of the `Pair` should be determined solely by the `key`
- Find the `index` of the `Pair` in the array of linked lists. This means that `table[index]` will be the linked list that we should put our `Pair`
- Iterate through the linked list just like we did in `ListMap` and check if any of the pairs in the linked list have the same `key`. If so, return the `value` of the pair. Otherwise, return `null`

```java
// SeparateChainingMap.java

import java.util.LinkedList;

public class SeparateChainingMap<K extends Comparable<? super K>, V>
implements Map<K, V> {
  private LinkedList<Pair<K, V>>[] table;

  @SuppressWarnings("unchecked")
  public SeparateChainingMap() {
    table = (LinkedList<Pair<K, V>>[]) new LinkedList[8];
    for (int i = 0; i < table.length; i++) {
      table[i] = new LinkedList<Pair<K, V>>();
    }
  }

  @Override
  public void put(K key, V value) {
    int hash = key.hashCode();
    int index = hash % table.length;
    if (index < 0) {
      index += table.length;
    }
    LinkedList<Pair<K, V>> list = table[index];

    for (Pair<K, V> pair : list) {
      if (pair.key.equals(key)) {
        pair.value = value;
        return;
      }
    }
    list.add(new Pair<K, V>(key, value));
  }

  @Override
  public V get(K key) {
    int hash = key.hashCode();
    int index = hash % table.length;
    if (index < 0) {
      index += table.length;
    }
    LinkedList<Pair<K, V>> list = table[index];

    for (Pair<K, V> pair : list) {
      if (pair.key.equals(key)) {
        return pair.value;
      }
    }
    return null;
  }
}
```