PA5 Generic HashMap

Due: Friday, November 4th, 11:59PM

Submission:

MyHashMap.java - a generic implementation of a map using a hash table.

Overview

A hash table is a data structure that maps keys to values by using a hash function to ‘hash’ the key into an index for an array. It supports constant time ‘puts’ and ‘gets’ and so it is a common way to implement ADTs such as maps/dictionaries and sets. This assignment involves the implementation of a generic map using a hash table.

Assignment

As you work on this assignment, consider the following:

- What are the time complexities for the methods?
- Are these time complexities always the case or can things get worse, and if so, why?
- What makes a good hash function?

Create a class called MyHashMap which implements a map using a hash table. Your MyHashMap class must have the methods listed in the documentation:
http://u.arizona.edu/~ccapriotti/MyHashMap/MyHashMap.html

The only special method is printTable which should output how many conflicts occur at each array slot (or bucket) and list the keys at that slot. Your hashTable should always have 8 slots. An example output for printTable is shown in the documentation.

It is recommended to use an ArrayList of linked lists. The linked lists can be the Java LinkedList class or your own linked list implementation where the key,value pairs link to each other, whatever you prefer.

Here is the code for the hash function. You should use this exact code in your project:

```
private int hash(K key) {
    int hashCode = key.hashCode();
    int index = hashCode % numBuckets;
    return Math.abs(index);
}
```

This hash map will use chaining to handle collisions. Your hash map should only have 8 slots. Collisions will occur.

Put the (key,value) pairs at the head of the linked list. If the same key is passed in with a different value, do the value update ‘in place’.

Methods from the documentation link:

`void`	`clear()`	Removes all of the mappings from this map.
`boolean`	`containsKey (K key)`	Returns true if this map contains a mapping for the specified key.
`boolean`	`containsValue (V val)`	Returns true if this map maps one or more keys to the specified value.
`V`	`get (K key)`	Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
`boolean`	`isEmpty()`	Returns true if this map contains no key-value mappings.
`java.util.Set<K>`	`keySet()`	Returns a Set view of the keys contained in this map.
`void`	`printTable()`	Outputs how many conflicts occur at each bucket and list the keys in that bucket.
`V`	`put (K key, V val)`	Associates the specified value with the specified key in this map.
`V`	`remove (K key)`	Removes the mapping for the specified key from this map if present.
`int`	`size()`	Returns the number of key-value mappings in this map.

public void printTable()

Outputs how many conflicts occur at each bucket and list the keys in that bucket.
Example output for this method:

```
Index 0: (0 conflicts), []
Index 1: (0 conflicts), []
Index 2: (0 conflicts), []
Index 3: (0 conflicts), []
Index 4: (0 conflicts), []
Index 5: (0 conflicts), [ExampleKeyX, ]
Index 6: (0 conflicts), [ExampleKeyY, ]
Index 7: (0 conflicts), []
Total # of conflicts: 0
```