Design a custom Hash Table ADT **not** based on the Java Collection Framework. Use format illustrated in the text.

* Description **done?**
* Attributes (fields)
  + number of buckets **done?**
* Operations(pre/post conditions, responsibilities, return value, exceptions)
  + hash function algorithm
  + collision resolution strategy **done?**
  + insertions and deletions **done?**
  + Searched **done?**
* Test plan

Turn hard copies of your documents on the due date.

Look to chapter 12 for help.

**Hash Table ADT**

**Description**

A hash table stores <key, value> pairs. A hash table is an indexable collection, each position in the hash table is called a bucket and can hold one or more entries. The entry keys are converted into an index that is used to access the bucket in the hash table. Given a key, the hash table provides the associated value. The type of the key is not specified, but it must be possible to test for equality among keys and values.

**Attributes**

size: The number of buckets containing entries in this table

entrySize: The number of entries stored in the table

buckets: The number of buckets in this table

loadFactor: entrySize/buckets

**Operations**

HashTable()

pre-condition: none

responsibilities: constructor – create an empty hash table

post-condition: size is set to 0, entrySize set to zero;

returns: nothing

insert (KeyType key, ValueType value)

pre-condition: key is not null

if key is in the hash table, value is not null

key can be compared for equality to other keys in the hash table

value can be compared for equality to other values in the hash table

responsibilities: inserts the <key, value> into the hash table. If the key already exits the value is replaced with the new value

post-condition: size is incremented by one if key did not exist, and bucket was empty

entrySize is incremented by one if key did not exist

loadFactor is altered accordingly

returns: null if key was not already in this hash table, the old value associated with the key otherwise

exception: if key is null, or key exists in table and value is null, or key and value cannot be compared to other keys/values in the table

delete(KeyType key)

pre-condition: key is not null, and can be compared for equality to other keys in this hash table

responsibilities: remove the key, and its associated value from this table

post-condition: size is decreased by one if key was found and the bucket is now empty.

entrySize is decreased by one if the key was found

returns: null if key was not found in table, the key otherwise

exception: if key is null or cannot be compared for equality to other keys in this table

get(KeyType key)

pre-condition: key and value are not null, and can be compared for equality to other keys and values in this hash table

responsibilities: get value associated with the key in this table

post-condition: table is unchanged

returns: null if key was not found in table, the associated value otherwise

exception: if key is null or cannot be compared for equality to other keys in this table.

containsValue(ValueType value)

pre-condition: value is not null, and can be compared for equality to other values in this hash table

responsibilities: determines if table contains an entry with this value

post-condition: table is unchanged

returns: true if value was found in table, false otherwise

exception: if value is null or cannot be compared for equality to other values in this table.

containsKey(KeyType key)

pre-condition: key is not null, and can be compared for equality to other keys in this hash table

responsibilities: determines if table contains an entry with this key

post-condition: table is unchanged

returns: true if key was found in table, false otherwise

exception: if key is null or cannot be compared for equality to other keys in this table

HashMap Test Plan

**Test Case 1**

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Purpose | Object state | Expected result |
| HashTable<k,v> table = new HashTable<k,v> (); | Create a new hash table object with no entries | Size: 0  entrySize: 0  loadFactor: 0 | Null |
| Table.isEmpty() | To verify empty table |  | True |
| Table.insert(k,v) | Put entry into table | Size: 1  entrySize: 1  loadFactor: 1/n | Null |
| Table.isEmpty() | To verify table is not empty |  | False |
| Table.containsKey (k) | Verify insert operation |  | True |
| Table.containsValue (v) | Verify insert operation |  | True |

**Test Case 2**

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Purpose | Object state | Expected result |
| HashTable<k,v> table = new HashTable<k,v> (); | Create a new hash table object with no entries | Size: 0  entrySize: 0  loadFactor: 0 | Null |
| Table.insert(k1,v1) | Put entry into table | Size: 1  entrySize: 1  loadFactor: 1/n  table(k1,v1) | Null |
| Table.insert(k1,v2) | Verify key exists, and replace its value | Size: 1  entrySize: 1  loadFactor: 1/n  table(k1,v1) | Null |
| Table.containsKey (k1) | Verify insert operation |  | True |
| Table.containsValue (v1) | Verify previous value replaced |  | False |
| Table.containsValue (v2) | Verify new value is in table |  | True |

**Test Case 3**

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Purpose | Object state | Expected result |
| HashTable<k,v> table = new HashTable<k,v> (); | Create a new hash table object with no entries | Size: 0  entrySize: 0  loadFactor: 0 | Null |
| Table.insert(k1,v1) | Put entry into table | Size: 1  entrySize: 1  loadFactor: 1/n  table(k1,v1) | Null |
| Table.insert(k2,v1) | Put entry with duplicate value into table | Size: 2  entrySize: 2  loadFactor: 2/n  table(k1,v1)(k2,v2) | Null |
| Table.containsKey (k1) | Verify insert operation |  | True |
| Table.containsKey (k2) | Verify insert operation |  | True |
| Object o1 = map.get(k1) | Verify value associated with k1 |  | V1 |
| Object o1 = map.get(k1) | Verify value associated with k1 |  | V1 |
| O1.equals(o2) | Verify both keys have the same value |  | True |