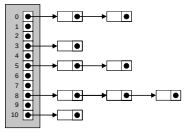

CSIS 3103

Ch 7: Hash Table Implementations and Applications

Interface KWHashMap

Method	Behavior
`V get(Object key)`	Returns the value associated with the specified `key`. Returns `null` if the `key` is not present.
`boolean isEmpty()`	Returns `true` if this table contains no `key-value` mappings.
`V put(K key, V value)`	Associates the specified `value` with the specified `key`. Returns the previous `value` associated with the specified `key`, or `null` if there was no mapping for the `key`.
`V remove(Object key)`	Removes the mapping for this `key` from this table if it is present (optional operation). Returns the previous `value` associated with the specified `key`, or `null` if there was no mapping.
`int size()`	Returns the size of the table.

Class HashTableOpen

Data Field	Attribute
`private Entry<K, V>[] table`	The hash table array.
`private static final int START_CAPACITY`	The initial capacity.
`private double LOAD_THRESHOLD`	The maximum load factor.
`private int numKeys`	The number of keys in the table excluding keys that were deleted.
`private int numDeletes`	The number of deleted keys.
`private final Entry<K, V> DELETED`	A special object to indicate that an entry has been deleted.

Data Field	Attribute
`private K key`	The key.
`private V value`	The value.
Constructor	**Behavior**
`public Entry(K key, V value)`	Constructs an `Entry` with the given values.

Method	Behavior
`public K getKey()`	Retrieves the key.
`public V getValue()`	Retrieves the value.
`public V setValue(V val)`	Sets the value.

Class HashTableOpen

Method	Behavior
`private int find(Object key)`	Returns the index of the specified key if present in the table; otherwise, returns the index of the first available slot.
`private void rehash()`	Doubles the capacity of the table and permanently removes deleted items.

Algorithm for `HashtableOpen.find(Object key)`

1. Set `index` to `key.hashCode() % table.length`.
2. if `index` < 0, add `table.length`.
3. while `table[index]` is not empty and the key is not at `table[index]`
4. increment `index`.
5. if `index` ≥ `table.length`
6. Set `index` to 0.
7. Return the `index`.

Class HashTableOpen

Algorithm for `HashtableOpen.get(Object key)`

1. Find the first table element that is empty or the table element that contains the key.
2. `if` the table element found contains the key
   return the value at this table element.
3. `else`
4. return `null`.

Algorithm for HashtableOpen.put(K key, V value)

1. Find the first table element that is empty or the table element that contains the key.
2. if an empty element was found
3. insert the new item and increment numKeys.
4. check for need to rehash.
5. return null.
6. The key was found. Replace the value associated with this table element and return the old value.

Class HashTableOpen

Algorithm for `HashtableOpen.remove(Object key)`

1. Find the first table element that is empty or the table element that contains the key.
2. if an empty element was found
3. return null.
4. Key was found. Remove this table element by setting it to reference DELETED,
increment numDeletes, and decrement numKeys.
5. Return the value associated with this key.

Algorithm for `HashtableOpen.rehash`

1. Allocate a new hash table that is double the size and has an odd length.
2. Reset the number of **keys** and number of deletions to 0.
3. Reinsert each table entry that has not been deleted in the new hash table.

Class HashTableChain

Data Field	Attribute
`private LinkedList<Entry<K, V>>[] table`	A table of references to linked lists of `Entry<K, V>` objects.
`private int numKeys`	The number of keys (entries) in the table.
`private static final int CAPACITY`	The size of the table.
`private static final int LOAD_THRESHOLD`	The maximum load factor.

Algorithm for `HashtableChain.get(Object key)`

1. Set `index` to `key.hashCode() % table.length.`
2. `if index` is negative
3. add `table.length`.
4. if `table[index]` is `null`
5. `key` is not in the table; return `null`.
6. For each element in the list at `table[index]`
7. `if` that element's key matches the search key
8. return that element's value.
9. `key` is not in the table; return `null`.

Class HashTableChain

Algorithm for `HashtableChain.put(K key, V value)`

1. Set index to `key.hashCode() % table.length.`
2. `if index` is negative, add `table.length.`
3. `if table[index]` is `null`
4. create a new linked list at `table[index].`
5. Search the list at `table[index]` to find the key.
6. `if` the search is successful
7. replace the value associated with this key.
8. return the old value.
9. `else`
10. insert the new key-value pair in the linked list at `table[index].`
11. increment `numKeys`.
12. `if` the load factor exceeds the `LOAD_THRESHOLD`
13. Rehash.
14. return `null`.

Class HashTableChain

Algorithm for `HashtableChain.remove(Object key)`

1. Set `index` to `key.hashCode() % table.length.`
2. `if index` is negative, add `table.length`.
3. `if table[index]` is `null`
4. `key` is not in the table; return `null`.
5. Search the list at `table[index]` to find the key.
6. `if` the search is successful
7. remove the entry with this key and decrement `numKeys`.
8. `if` the list at `table[index]` is empty
9. Set `table[index]` to `null`.
10. return the value associated with this key.
11. The `key` is not in the table; return `null`.

Methods hashCode and equals

`Object.hashCode` calculates an object's hash code based on its address, not its contents

Most predefined classes override `hashcode`

- Java recommends also overriding `hashCode` if you override the `equals` method,
  - use the same data field(s) as in `equals` method

```
if obj.equals(obj2) is true,
then obj1.hashCode == obj2.hashCode
```

Cell Phone Contact List

Problem

A cell phone manufacturer wants a program to maintain contact lists on their phones

The manufacturer has provided the interface:

Method	Behavior
`List<String> addOrChangeEntry(String name, List<String> numbers)`	Changes the numbers associated with the given `name` or adds a new entry with this `name` and list of numbers. Returns the old list of numbers or `null` if this is a new entry.
`List<String> lookupEntry(String name)`	Searches the contact list for the given `name` and returns its list of numbers or `null` if the `name` is not found.
`List<String> removeEntry(String name)`	Removes the entry with the specified `name` from the contact list and returns its list of numbers or `null` if the `name` is not in the contact list.
`void display();`	Displays the contact list in order by `name`.

Cell Phone Contact List

Analysis

- A map will associate the name (key) with a list of phone numbers (value)
- Implement `ContactListInterface` by using a `Map<String, List<String>>` object for the data type