Intro to Hashmaps

What is a Hash Map?

A Hash Map (also called a “Hash Table”) is a type of collection that has “Key, Value pairs” and we use the key as a way to map where to place these pairs in memory. In general, Hash Maps and Hash Tables are interchangeable terminology, but it is important to note that, in Java, these two are subtly different. HashMaps are not “synchronized” which means they are not thread safe for concurrent applications. HashTables are thread safe.

The term “hash” comes from the fact that we are using a type of array behind the scenes to store the data structure and we must convert the key to an array index using some kind of mathematical calculation. This process of acquiring an array index is called “hashing.”

Also of note is that Hash maps do not contain sorted values and should never be relied upon to. If you need a sorted data structure, consider something self-sorted like a Binary Search Tree.

Furthermore, realize that Hash maps should not allow duplicate keys, but duplicate values are ok. Think of trying to write a value to an array at the same index.

Purpose of Hashmaps

What is the purpose for them?

When we deal with a pure array, we get constant performance (O(1)) for random access but have expensive add/remove when we need to resize the array.

Linked lists, on the other hand, give us good performance for add/remove but require O(n) time (worst case) for access when we allow arbitrary access to the nodes.

A Hash map gives us the opportunity to use more space complexity in an attempt to gain better performance with time complexity. Generally speaking, we should be able to get constant time add and close to constant time for delete and retrieval of values given the key.

Another neat use for Hash Maps is that we can use non-numeric keys to access data values. With an array, we are locked into using an index in the range of 0 to n-1 but with a hash map, we can use a String for a key even (which hashes to an array index behind the scene) but it gives us a more “human” way to access data as opposed to recalling index numbers.

Java Implementation

Does Java provide a default HashMap class?

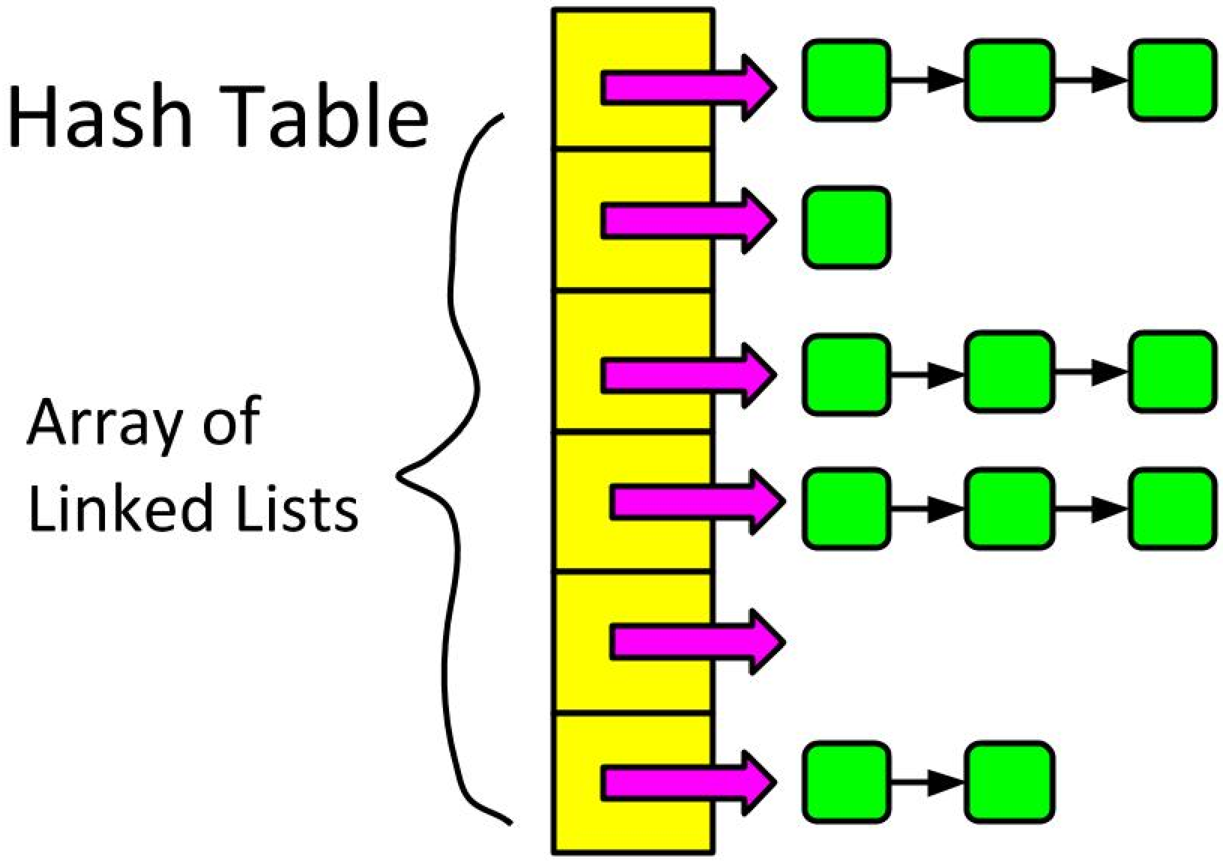
Yes, Java has a class called “HashMap”. This class allows generic types for both Key and Values. Here is an example of how we can use this class:

```
public static void main(String[] args){

        // For the API...

        HashMap<String, String> map = new HashMap<>();

        map.put("Tester", "Hello!");

        map.put("Testy", "Howdy!");

        map.put("Test", "Hola!");

        p(map.toString());

        p("Deleting Tester...");

        map.remove("Tester");

        p(map.toString());

        p("Clearing the map...");

        map.clear();

        p(map.toString());

    }
```

Which Data Structure?

What data structure should we use to store a HashMap behind-the-scene?

The classic way that Hash maps were stored is by using an array of linked lists. This was clever since it was an attempt to obtain the strengths of both data structures with none of their weaknesses (except additional space complexity).



As of Java 8, Java uses an array of an implementation of a balanced binary tree for the Hash Map. The benefit of this, in the developers' minds, is that lookups of values inside of the "buckets" can occur in worst case O(log n) instead of O(n). The trade-off is that performance may be lower than a linked list when there are few values in the bucket due to the cost associated with balancing the tree and maintaining a sorted structure.

For our custom implementation, we are going to use the classic model of an array of linked lists. We will wrap our linked lists in a class called "HashChain" for simplicity of writing the Hash table class; as well as supporting special functionality such as a unique toString method.

Class Functionality

What functionality should our HashMap class contain?

Our HashMap class should contain the following methods:

```
int hash(String key);                    // This is our private hashing method which acquires the array index
String getValue(String key);             // This looks up the value based on the key
                                          // (We are using Strings for fun for both)
void put(String key, String value);      // This adds the key, value pair to the table
void delete(String key);                 // This removes the key, value pair from the table
boolean isEmpty();                        // Let’s us know if there is nothing in the table
int size();                               // Returns number of key, value pairs in table
void clear();                             // Resets the table to empty
String toString();                       // Prints the key,value pairs in a unique way
```

Implement HashChain class

How do we write the code from scratch?

First, we write a helper class for the linked list “buckets” we are going to use…we call this “HashChain”:

```
package Main;

import java.util.LinkedList;

public class HashChain{

    // Fields

    private LinkedList<Node> chain;

    private class Node{

        String key;

        String value;

        public Node(String key, String value){

            this.key = key;

            this.value = value;

        }

    }

    // Constructor

    public HashChain(){

        chain = new LinkedList<Node>();

    }

    // Methods

    public boolean addNode(String key, String value){

        if(contains(key))

            return false;       // Do not add duplicate key!

                                // (We could also choose to replace existing key)

        Node node = new Node(key, value);

        chain.add(node);

        return true;

    }

    public boolean contains(String key){

        for(int i = 0; i < chain.size(); i++)

            if(chain.get(i).key.equals(key))
```

```
                return true;
            return false;
        }
        public String getValue(String key){
            for(int i = 0; i < chain.size(); i++)
                if(chain.get(i).key.equals(key))
                    return chain.get(i).value;
            return null;
        }
        public boolean removeNode(String key){
            for(int i = 0; i < chain.size(); i++)
                if(chain.get(i).key.equals(key)){
                    chain.remove(i);
                    return true;
                }
            return false;
        }
        public void clear(){
            chain.clear();
        }
        public String toString(){
            String ret = "[";
            for(int i = 0; i < chain.size(); i++)
                ret += "<" + chain.get(i).key + ", " + chain.get(i).value + ">,";
            if(ret.length() > 1)      // Trim trailing comma only if we have an item in here
                ret = ret.substring(0, ret.length()-1);
            ret += "]";
            return ret;
        }
    }
```

Implement Custom Hash map class

Now we write the actual Hash map which we call “Hash”:

/* HashMap routine by: Matthew W. Phillips 2020

- I didn't use a resize method because HashMaps benefit from the largest possible
- range of indexes from the beginning so resizing is not needed in our case.
- This implementation uses an array of type "HashChain" which is a wrapper for
- a LinkedList with our Key,Value nodes as the type */

```
package Main;

public class Hash{

    // Fields

    private final int M = 8;  // Size of array

    private HashChain[] map;

    private int n;                    // How many key,value pairs in map

    // Constructor

    public Hash(){

        map = new HashChain[M];

        for(int i = 0; i < M; i++)

            map[i] = new HashChain();           // Initialize

        n = 0;

    }

    // Methods

    private int hash(String key){

        final int R = 31;

        int h = 0;

        for(int i = 0; i < key.length(); i++)

            h = ((R * h) + key.charAt(i)) % M;      // Text recommends 31 as 'R'

        return h;

    }

    public String getValue(String key){

        int e = hash(key);

        return map[e].getValue(key);

    }

    public void put(String key, String value){
        int e = hash(key);
        if(map[e].addNode(key, value))
            n++;
    }
    public void delete(String key){
        int e = hash(key);
        if(map[e].removeNode(key))
            n--;
    }
    public boolean isEmpty(){
        return n == 0;
    }
    public int size(){
        return n;
    }
    public void clear(){
        for(int i = 0; i < M; i++)
            map[i].clear();
        n = 0;
    }
    public String toString(){
        String ret = "";
        for(int i = 0; i < M; i++)
            ret += map[i].toString() + "\n";
        return ret;
    }
}
```

Example Usage of Custom class

How would we use this code in the main method?

Below is an example of how we can use our Hash class code:

```
public class HashTest{

    public static void main(String[] args){

        Hash hash = new Hash();

        hash.put("Tester", "Hello!");

        hash.put("Testy", "Howdy!");

        hash.put("Test", "Hola!");

        p("Value of 'Test' is: " + hash.getValue("Test"));

        hash.delete("Test");

        hash.clear();

        p(hash.toString());          // View the Hashmap

        p(hash.isEmpty());

    }
```