How HashMap Works

# Example:

Put operations…

Graphical user interface

Description automatically generated with medium confidence

We are trying to put the key King and value 100 into the HashMap. The put API is called and the put API basically computes hash of the key, which is hash of King, which is in this case 2306996. We cannot have an array of this size in Java. Huge arrays will make you run out of memory and cause all kinds of other issues. That is why we use HashMaps to truncate the size of the array. Afterwards you must run a little index computation to find out where exactly we can fit this hash code in range 0 to 15. The HashMap implementation uses the bitwise operation as shown above. We can see that the index computes to 4. That means that the entry will go into the index of 4 as a node. In the picture above you can see that a new node is created with the key value, the hash value, the value itself which is 100(which is a score) and null meaning is not pointing to any other node (KING | 2306996 | 100 | null).

# Collision

In theory as shown below we have some sort of a collision because we already have an entry at index of 4. This new entry which also is stored in index of 4 will be added as the next node of the already existing node at index 4. So that means the pointer in this node which is basically created for Kind will point to the new node that is created for Blake. Whenever there is a collision theoretically if multiple hash ports are computing the same index, those entries will be stored as a linked list of nodes. Also, something to note HashMap allows you to store **null** as a key.

Graphical user interface, application

Description automatically generated

# Get Operations

A picture containing graphical user interface

Description automatically generated

We want to get the score that Clark got in the game. In order to do that we call the get operation scores.get(“CLARK”);. The get operation does the same set of operations, it finds hash of the key, which is this number 64205538, then it computes the index where that key could have been fitted, in this case its index of 2. We can look up an index 2 in out table where we can see an entry. With that we compare the hash code of the key against the hash code available there on that entry, that matches. Following we compare the key itself against the key that is available using the equals method (Clark -> Clark), which also matches. We found the match. The value at that node is returned to the caller and the caller gets the score which is 90.

# In linked lists:

In linked lists the operation goes through every entry and proves hash code for matches, if the entry does not match it moves on to the next entry until it finds a match for two of the same hash codes for example hash code = 2656862 matches the entry hash code which holds the same value of 2656862. Once the match has been found it caller gets the value of the score.

A picture containing graphical user interface

Description automatically generated

Also check out the related file over here.