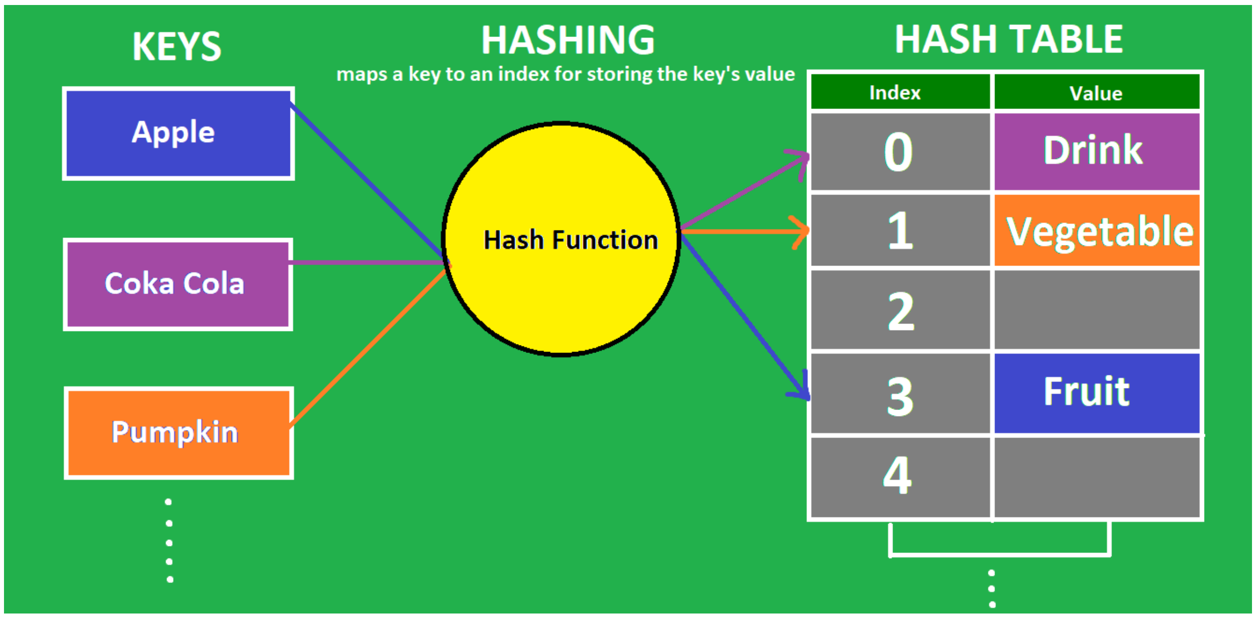
Hashing Techniques in Java

The hash function is a key-value mapping function. When two or more keys are mapped to the same value using these hashing methods, there exists duplicate values. The use of chain hashing prevents collisions. Each hash table cell should lead to a linked list of entries that have the same hash function value.

Let's create a hash function that returns a hash table with 'N' buckets.



To add a node to the hash table, we need to find the hash index for the specified key.

1. hashIndex = key % noOfBuckets

The hash function might also be used to figure it out.

Insert: Go to the bucket that corresponds to the hash index determined above and add the new node to the end of the list.

Delete: To remove a node from a hash table, compute the key's hash index, move to the bucket that corresponds to the calculated hash index, search the list in the current bucket for the node with the supplied key, and remove it (if found).

Methods for Implementing hashing in Java

1. HashTable-based Method(A synchronised implementation of hashing)

HashTableDemo.java

```
import java.util.*;
public class HashTableDemo
{
    /* Driver Code */
    public static void main(String args[])
    {
      /* Create a HashTable to store String values corresponding to integer keys */
       Hashtable<Integer, String> hm = new Hashtable<Integer, String>();
       /* Input the values */
       hm.put(3, "You are visiting");
       hm.put(5, "Hello");
       hm.put(1, "website");
       hm.put(2, "Javatpoint");
       /* Display the Hashtable */
       System.out.println(hm);
    }
}
```

Output:

```
{5=Hello, 3=You are visiting, 2=Javatpoint, 1=website}
```

Here, the above Java code demonstrates the use of Hashtable.

2. HashMap-based Method (A non-synchronized faster implementation of hashing)

HashMapDemo.java

```java
import java.util.*;
public class HashMapDemo
{
    /* Function to create HashMap from array */
    static void HashMapCreation(int arr[])
    {
        /* Creates an empty HashMap */
        HashMap<Integer, Integer> hashmap = new HashMap<Integer, Integer>();
        for (int i = 0; i < arr.length; i++)
        {
            /* Get if the element is present */
            Integer n = hashmap.get(arr[i]);
            /* If this is first occurrence of element Insert the element */
            if (hashmap.get(arr[i]) == null)
            {
                hashmap.put(arr[i], 1);
            }
            /* If element is already present in hash map Increment the count of
element by 1 */
            else
            {
                hashmap.put(arr[i], ++n);
            }
        }
        /* Display HashMap */
        System.out.println(hashmap);
    }
    /* Driver Code */
    public static void main(String[] args)
    {
        int arr[] = { 1, 6, 5, 10, 6, 6, 10 };
        HashMapCreation(arr);
    }
}
```

Output:

```
{1=1, 5=1, 6=3, 10=2}
```

The above Java code displays the use of Hashmap. It is a non-synchronized faster method for hashing.

3. LinkedHashMap-based method(Similar to HashMap, but keeps order of elements)

LinkedHashMapDemo.java

```
import java.util.*;
public class LinkedHashMapDemo
{
    /* Driver Code */
    public static void main(String arg[])
    {
        /* Creates a Linked Hashmap */
        LinkedHashMap<String, String> lhm = new LinkedHashMap<String, String>();
        /* Enter key values to the Hashmap */
        lhm.put("One", "Robin");
        lhm.put("Two", "Satyam");
        lhm.put("Three", "Kanishk");
        /* Prints the elements of Hashmap in an order as they are entered */
        System.out.println(lhm);
        System.out.println("Getting value for key 'one': " + lhm.get("One"));
        System.out.println("Size of the Hashmap: " + lhm.size());
        System.out.println("Is Hashmap empty? " + lhm.isEmpty());
        System.out.println("Contains key 'two'? "+  lhm.containsKey("Two"));
        System.out.println("Contains value 'Kanishk? "+
    lhm.containsValue("Kanishk"));
        System.out.println("delete element 'one': " +  lhm.remove("One"));
        System.out.println(lhm);
    }
}
```

Output:

```
{One=Robin, Two=Satyam, Three=Kanishk}
Getting value for key 'one': Robin
Size of the Hashmap: 3
Is Hashmap empty? false
Contains key 'two'? true
Contains value 'Kanishk? true
delete element 'one': Robin
{Two=Satyam, Three=Kanishk}
```

In the above program LinkedHashMap is used with its different methods such as ***size(), get(), isEmpty(), containsKey(), containsValue(), remove().***

4. ConcurretHashMap-based Method (Similar to HashTable, Synchronized, but faster as multiple locks are used)

DemoforConcurrentHashMap.java

```
import java.util.concurrent.*;
```

```java
public class DemoforConcurrentHashMap
{
    /* Driver Code */
    public static void main(String[] args)
    {
        /* Creates a ConcurrentHashMap */
        ConcurrentHashMap<Integer, String> ch = new ConcurrentHashMap<Integer, String>();
        ch.put(201, "How");
        ch.put(202, "are");
        ch.put(203, "you");
        /* Display the elements of ConcurrentHashMap */
        System.out.println("ConcurrentHashMap: " + ch);
        /* add 'How' at 202 key. As it is already present in ConcurrentHashMap object use putIfAbsent method */
        ch.putIfAbsent(202, "How");
        /* Display the elements of ConcurrentHashMap */
        System.out.println("\nConcurrentHashMap: " + ch);
        /* replace the value for key 201 from "How" to "Who" */
        ch.replace(201, "How", "Who");
        /* Display the elements of ConcurrentHashMap */
        System.out.println("\nConcurrentHashMap: " + ch);
        /* Try to remove entry for 203 key as it is present */
        ch.remove(203, "you");
        /* Display the elements of ConcurrentHashMap */
        System.out.println("\nConcurrentHashMap: " + ch);
    }
}
```

Output:

```
ConcurentHashMap: {201=How, 202=are, 203=you}
ConcurentHashMap: {201=How, 202=are, 203=you}
ConcurentHashMap: {201=Who, 202=are, 203=you}
ConcurentHashMap: {201=Who, 202=are}
```

The above Java program displays an implementation of ConcurrentHashMap with its different methods such as ***put(), putIfAbsent(), replace(), remove().***