```
public class Hash<K, V> {
    private HashNode[] buckets;
    private int size;
    private int numberOfBucket;

    public Hash() {
        this(10);
    }

    public Hash(int capacity) {
        this.size = 0;
        this.numberOfBucket = capacity;
        this.buckets = new HashNode[this.numberOfBucket];
    }

    private static class HashNode<K, V> {
        private K key;
        private V value;
        private HashNode<K, V> next;

        public HashNode(K key, V value) {
            this.key = key;
            this.value = value;
            this.next = null;
        }
    }

    public void put(K key, V value) {

        if (null == key || null == value) {
            throw new IllegalArgumentException("Key or Value is null");
        }

        int bucketIndex = getIndex(key);
        HashNode<K, V> head = buckets[bucketIndex];
        while (null != head) {
            if (head.key.equals(key)) {
                head.value = value;
                return;
            }
            head = head.next;
        }
        head = buckets[bucketIndex];
        HashNode<K, V> newNode = new HashNode<>(key, value);
        newNode.next = head;
        buckets[bucketIndex] = newNode;
        this.size++;
    }

    private int getIndex(K key) {
        return (int) key % this.numberOfBucket;
    }

    public V get(K key) {
        int bucketIndex = getIndex(key);
        HashNode<K, V> head = this.buckets[bucketIndex];
        while (null != head) {
            if (head.key.equals(key)) {
                return head.value;
            }
            head = head.next;
        }
        return null;
    }
```

```
    public void remove(K key) {
        if (null == key) {
            throw new IllegalArgumentException("Key  is null");
        }

        int bucketIndex = getIndex(key);
        HashNode<K, V> head = this.buckets[bucketIndex];
        HashNode<K, V> previous = null;
        while (null != head) {
            if (head.key.equals(key)) {
                if (null != previous) {
                    previous.next = head.next;
                } else {
                    this.buckets[bucketIndex] = head.next;
                }
                this.size--;
                break;
            }
            previous=head;
            head = head.next;
        }

    }

    public boolean containsKey(K key) {
        int bucketIndex = getIndex(key);
        HashNode<K, V> head = this.buckets[bucketIndex];
        while (null != head) {
            if (head.key.equals(key)) {
                return true;
            }
            head = head.next;
        }
        return false;
    }

    public int getSize() {
        return this.size;
    }

    public boolean isEmpty() {
        return this.size == 0;
    }

    public static void main(String[] args) {
        Hash<Integer, String> hash = new Hash<>();
        hash.put(1, "A");
        hash.put(11, "B");
        hash.put(101, "C");

        System.out.println(hash.get(1));
        System.out.println(hash.get(11));
        System.out.println(hash.get(101));

        hash.remove(11);
        hash.remove(1);
        hash.remove(101);

    }
}
```

What is Hashing

It is the process of converting an object into an integer value. The integer value helps in indexing and faster searches.

What is HashMap

HashMap is a part of the Java collection framework. It uses a technique called Hashing. It implements the map interface. It stores the data in the pair of Key and Value. HashMap contains an array of the nodes, and the node is represented as a class. It uses an array and LinkedList data structure internally for storing Key and Value. There are four fields in HashMap.

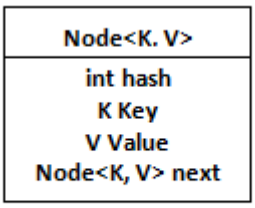


Figure: Representation of a Node

Before understanding the internal working of HashMap, you must be aware of hashCode() and equals() method.

- **equals():** It checks the equality of two objects. It compares the Key, whether they are equal or not. It is a method of the Object class. It can be overridden. If you override the equals() method, then it is mandatory to override the hashCode() method.
- **hashCode():** This is the method of the object class. It returns the memory reference of the object in integer form. The value received from the method is used as the bucket number. The bucket number is the address of the element inside the map. Hash code of null Key is 0.
- **Buckets:** Array of the node is called buckets. Each node has a data structure like a LinkedList. More than one node can share the same bucket. It may be different in capacity.

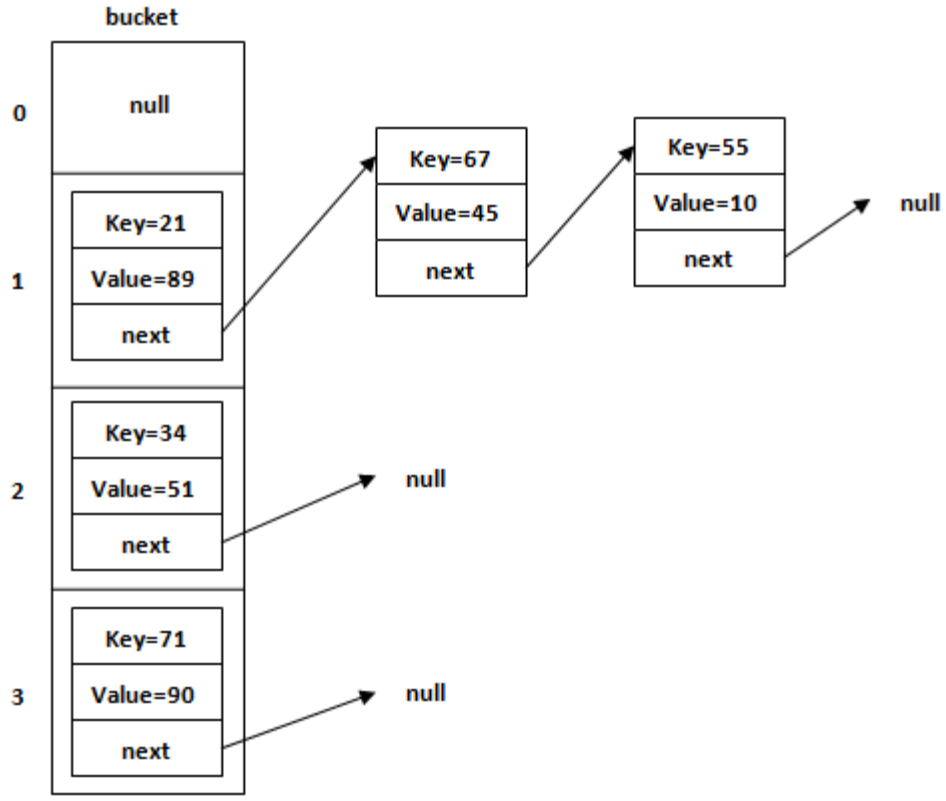


Figure: Allocation of nodes in Bucket

Insert Key, Value pair in HashMap

We use put() method to insert the Key and Value pair in the HashMap. The default size of HashMap is 16 (0 to 15).

Example

In the following example, we want to insert three (Key, Value) pair in the HashMap.

```
1. HashMap<String, Integer> map = new HashMap<>();
2. map.put("Aman", 19);
3. map.put("Sunny", 29);
4. map.put("Ritesh", 39);
```

Let's see at which index the Key, value pair will be saved into HashMap. When we call the put() method, then it calculates the hash code of the Key "Aman." Suppose the hash code of "Aman" is 2657860. To store the Key in memory, we have to calculate the index.

Calculating Index

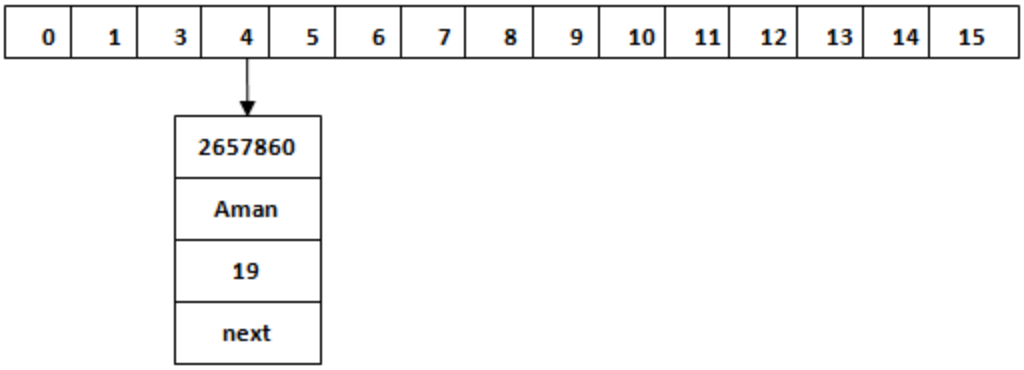
Index minimizes the size of the array. The Formula for calculating the index is:

1. Index = hashcode(Key) & (n-1)

Where n is the size of the array. Hence the index value for "Aman" is:

1. Index = 2657860 & (16-1) = 4

The value 4 is the computed index value where the Key and value will store in HashMap.

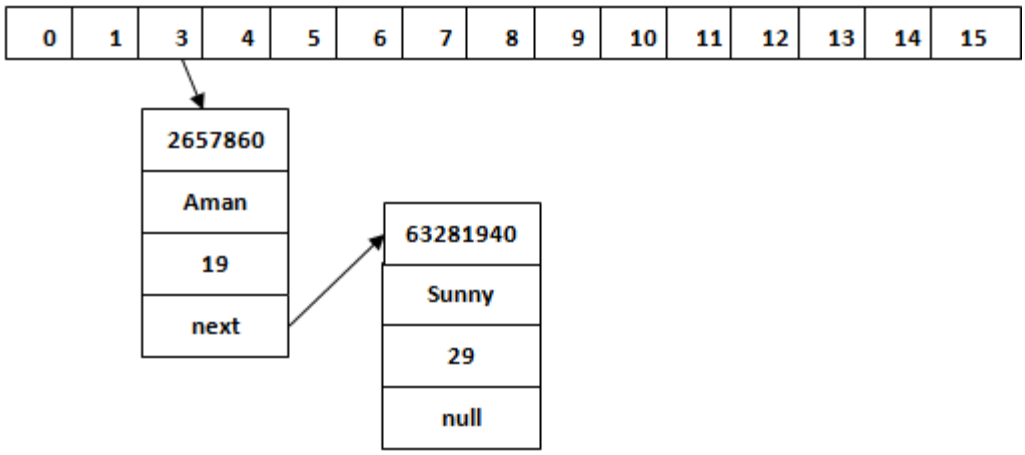


Hash Collision

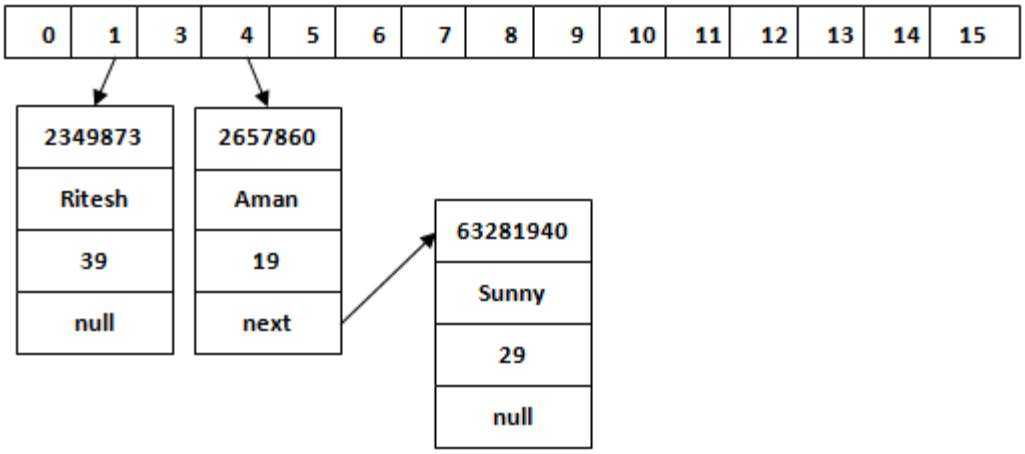
This is the case when the calculated index value is the same for two or more Keys. Let's calculate the hash code for another Key "Sunny." Suppose the hash code for "Sunny" is 63281940. To store the Key in the memory, we have to calculate index by using the index formula.

1. Index=63281940 & (16-1) = 4

The value 4 is the computed index value where the Key will be stored in HashMap. In this case, equals() method check that both Keys are equal or not. If Keys are same, replace the value with the current value. Otherwise, connect this node object to the existing node object through the LinkedList. Hence both Keys will be stored at index 4.



Similarly, we will store the Key "Ritesh." Suppose hash code for the Key is 2349873. The index value will be 1. Hence this Key will be stored at index 1.



get() method in HashMap

get() method is used to get the value by its Key. It will not fetch the value if you don't know the Key. When get(K Key) method is called, it calculates the hash code of the Key.

Suppose we have to fetch the Key "Aman." The following method will be called.

1. map.get(**new** Key("Aman"));

It generates the hash code as 2657860. Now calculate the index value of 2657860 by using index formula. The index value will be 4, as we have calculated above. get() method search for the index value 4. It compares the first element Key with the given Key. If both keys are equal, then it returns the value else check for the next element in the node if it exists. In our scenario, it is found as the first element of the node and return the value 19.

Let's fetch another Key "Sunny."

The hash code of the Key "Sunny" is 63281940. The calculated index value of 63281940 is 4, as we have calculated for put() method. Go to index 4 of the array and compare the first element's Key with the given Key. It also compares Keys. In our scenario, the given Key is the second element, and the next of the node is null. It compares the second element Key with the specified Key and returns the value 29. It returns null if the next of the node is null.

```
public class MyHashMap {

    private HashNode[] bucket;
    private int size;
    private int numberOfBuckets;

    public MyHashMap() {
        this(10);
    }

    public MyHashMap(int capacity) {
        this.size = 0;
        this.numberOfBuckets = capacity;
        this.bucket = new HashNode[this.numberOfBuckets];
    }


    public void put(Integer key, String value) {

        if (null == key || null == value) {
            throw new IllegalArgumentException("key or value should not be null");
        }

        int bucketIndex =getIndex(key);
        HashNode head = bucket[bucketIndex];
        while(null!=head){
            if(head.key==key){
                head.value=value;
                return;
            }
            head = head.next;
        }
        head = bucket[bucketIndex];
        this.size++;
        HashNode newNode = new HashNode(key, value);
        newNode.next= head;
        bucket[bucketIndex] = newNode;
    }

    public String get(Integer key){

        int bucketIndex = getIndex(key);
        HashNode head = this.bucket[bucketIndex];
        while(null!=head){
            if(head.key == key){
                    return head.value;
            }
            head = head.next;
        }

        return null;
    }


    private int getIndex(Integer key) {
        return key % this.numberOfBuckets;
    }
```

```
    public int getSize() {
        return this.size;
    }

    public boolean isEmpty() {
        return this.size == 0;
    }


    public class HashNode {

        private Integer key;//can be generic type
        private String value; // can be generic Type
        private HashNode next;

        public HashNode(Integer key, String value) {
            this.key = key;
            this.value = value;
            this.next = null;
        }
    }

    public static void main(String[] args) {
        MyHashMap map = new MyHashMap();
        map.put(1, "a");
        map.put(11, "abc");
        map.put(21, "abcc");
        map.put(11, "updated value");

        String value = map.get(31);
        System.out.println(value);
    }
}

//TODO

Add remove
And containsKey method.
```