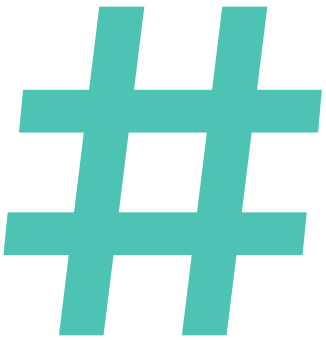


# Hash Table

## Step 01 What is a Hash Table

Imagine you have a list of phone numbers that you need to store and manage efficiently. by implementing a hash table in this scenario, you can associate each person's name with their corresponding phone number, where the person's name is the key and the phone number is a value. This allows for quick and direct access to phone numbers based on the names of the individuals.

Hash table is a type of data structure in which the index value of the data element is generated from a hash function hash = k % array_size.

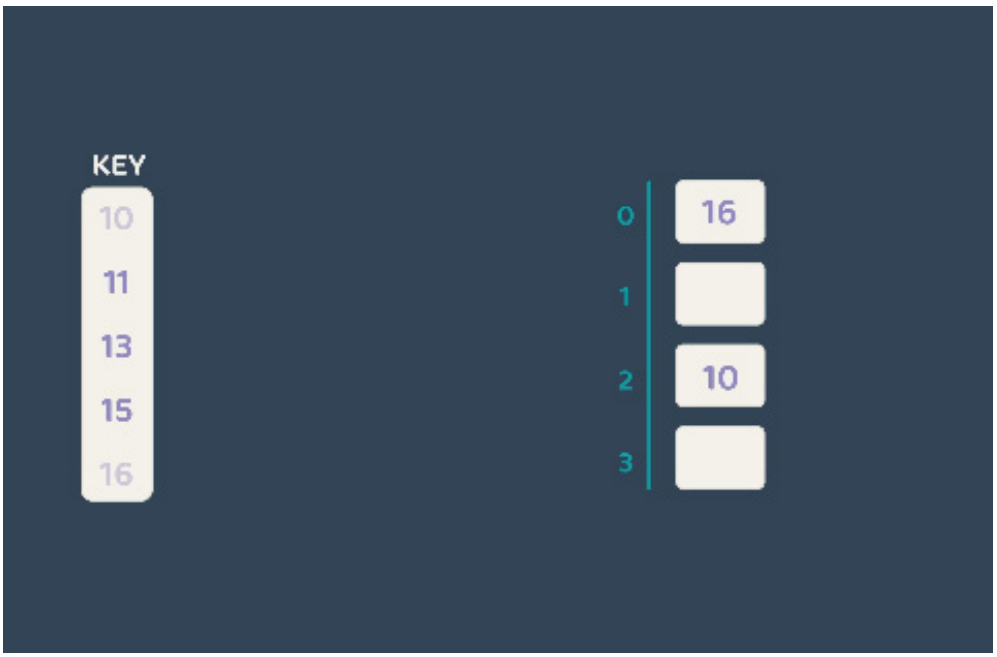
```
01  public int hashFunction(int key, int arrSize) {
02        return key % arrSize;
03    }
```

## Step 02 Collision Handling

Collision Handling happens when two keys are hashed to the same index in a hash table. Collisions are a problem because every slot in a hash table is supposed to store a single element, to handle collisions there are various techniques to handle such as chaining and open addressing.

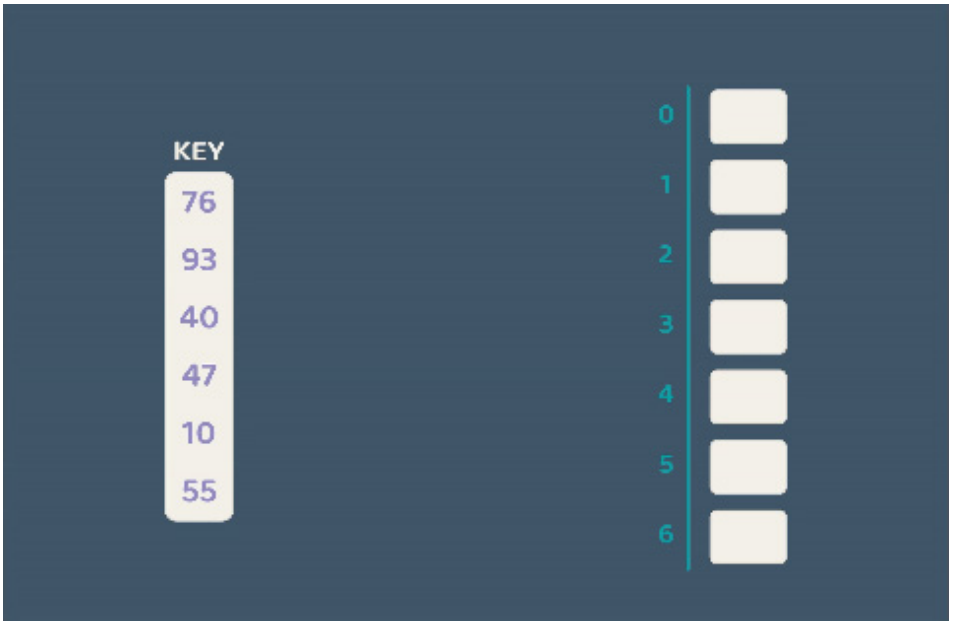
### 01 Chaining

> Each bucket in the hash table contains a linked list to store multiple key-value pairs that hash to the same index so we can handle a random number of collisions using an array of linked lists, each index has its own linked list



## 02 Open Address

Stores all key-value pairs directly in the hash table itself, without using separate data structures like linked lists. when a collision occurs, open addressing involves probing the table to find an alternative index for the colliding element (checking the next available slot).



## Step 03 Implementation

```
 private static class Node {
    int key;
    int value;
    Node next;
    public Node(int key, int value) {
        this.key = key;
        this.value = value;
    }
}
```

CODE Create Node class.

```
public class HashTable {
    private int size;
    private Node[] table;
    public HashTable(int size) {
        this.size = size;
        table = new Node[size];
    }
```

CODE Create a hash table using the linked lists collision chaining technique.

```
public void put(int key, int value) {
    int index = hash(key);
    Node node = table[index];
    if (node == null) {
        table[index] = new Node(key, value);
    } else {
        while (node.next != null) {
            node = node.next;
        }
        node.next = new Node(key, value);
    }
}
```

CODE The put method inserts a new key-value pair into the hash table.

```
public int get(int key) {
    int index = hash(key);
    Node node = table[index];
    while (node != null) {
        if (node.key == key) {
            return node.value;
        }
        node = node.next;
    }
    return -1;
 // not found
}
```

CODE The get method retrieves the value associated with a given key.

```
private int hash(int key) {
        return key % size;
    }
```

CODE The hash method calculates the index of the table based on the key.

```
   public static void main(String[] args) {
        HashTable hashTable = new HashTable(10);

        hashTable.put(1, 10);
        hashTable.put(2, 20);
        hashTable.put(11, 30);
        hashTable.put(12, 40);


        System.out.println("Get value for key 1: " + hashTable.get(1));
        System.out.println("Get value for key 2: " + hashTable.get(2));
        System.out.println("Get value for key 11: " + hashTable.get(11));
        System.out.println("Get value for key 12: " + hashTable.get(12));
        System.out.println("Get value for key 3: " + hashTable.get(3));


    }
}
```

CODE Main method.

OUTPUT

Get value for key 1: 10

Get value for key 2: 20

Get value for key 11: 30

Get value for key 12: 40

Get value for key 3: -1