Practical -21

Aim:- . Implement a hash table using separate chaining for collision handling. Perform operations like insertion, deletion, and search on the hash table.

Flow of the Program

1. **Creating the Hash Table**:
   - The **HashTable** constructor is called with a capacity of 10.
   - An array of linked lists is created, where each index can hold a linked list of **HashNode** objects.
2. **Inserting Key-Value Pairs**:
   - **Insert "Alice", 25**:
     - The hash function computes the index for "Alice".
     - The linked list at that index is checked. Since it's empty, a new **HashNode** is created and added.
   - **Insert "Bob", 30**:
     - The hash function computes the index for "Bob".
     - A new **HashNode** is created and added to the linked list at that index.
   - **Insert "Charlie", 35**:
     - The hash function computes the index for "Charlie".
     - A new **HashNode** is created and added to the linked list at that index.
3. **Searching for Values**:
   - **Search for "Alice"**:
     - The hash function computes the index for "Alice".
     - The linked list at that index is traversed, and the value 25 is found.
   - **Search for "Bob"**:
     - The hash function computes the index for "Bob".
     - The value 30 is found in the linked list.
   - **Search for "Charlie"**:

        - The hash function computes the index for "Charlie".
        - The value 35 is found in the linked list.
    - **Search for "David"**:
        - The hash function computes the index for "David".
        - The linked list is traversed, but no entry is found, so **null** is returned.

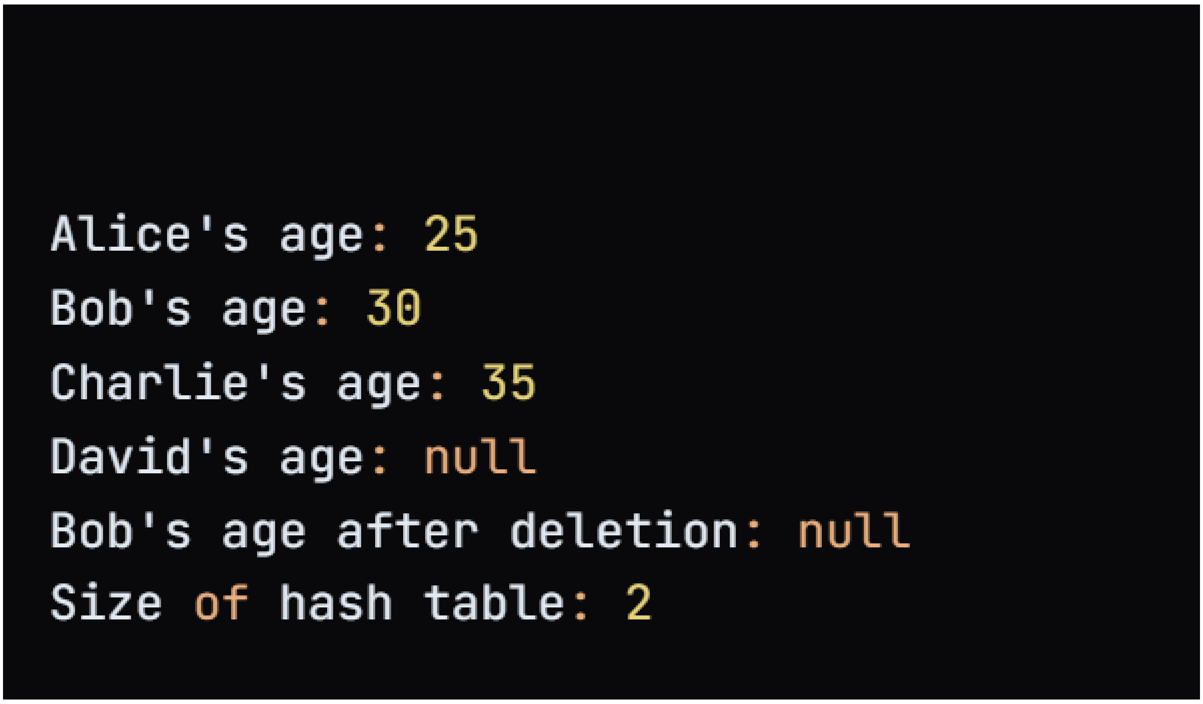
4. **Deleting a Key**:
    - **Delete "Bob"**:

Progaram:-

```
import java.util.LinkedList;


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


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


    private LinkedList<HashNode<K, V>>[] table;
    private int capacity;
    private int size;

    @SuppressWarnings("unchecked")
    public HashTable(int capacity) {
        this.capacity = capacity;
        this.size = 0;
        table = new LinkedList[capacity];
        for (int i = 0; i < capacity; i++) {
            table[i] = new LinkedList<>();
        }
    }

    private int hash(K key) {
        return Math.abs(key.hashCode()) % capacity;
    }

    public void insert(K key, V value) {
        int index = hash(key);
        LinkedList<HashNode<K, V>> bucket = table[index];

        for (HashNode<K, V> node : bucket) {
            if (node.key.equals(key)) {
                node.value = value; // Update existing value
                return;
            }
        }

        bucket.add(new HashNode<>(key, value));
        size++;
    }

    public V search(K key) {
        int index = hash(key);
        LinkedList<HashNode<K, V>> bucket = table[index];

        for (HashNode<K, V> node : bucket) {
            if (node.key.equals(key)) {
                return node.value; // Return the value if found
            }
        }
        return null; // Key not found
    }

    public void delete(K key) {
        int index = hash(key);
        LinkedList<HashNode<K, V>> bucket = table[index];
        for (HashNode<K, V> node : bucket) {
            if (node.key.equals(key)) {
                bucket.remove(node);
                size--;
                return;
            }
        }
    }

    public int size() {
        return size;
    }

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

    public static void main(String[] args) {
        HashTable<String, Integer> hashTable = new HashTable<>(10);

        // Inserting key-value pairs
        hashTable.insert("Alice", 25);
        hashTable.insert("Bob", 30);
        hashTable.insert("Charlie", 35);
```

```
        // Searching for values
        System.out.println("Alice's age: " + hashTable.search("Alice")); // Output: 25
        System.out.println("Bob's age: " + hashTable.search("Bob")); // Output: 30
        System.out.println("Charlie's age: " + hashTable.search("Charlie")); // Output: 35
        System.out.println("David's age: " + hashTable.search("David")); // Output: null

        // Deleting a key
        hashTable.delete("Bob");
        System.out.println("Bob's age after deletion: " + hashTable.search("Bob")); // Output: null

        // Checking size
        System.out.println("Size of hash table: " + hashTable.size()); // Output: 2
    }
}
```

Output:-



```
Alice's age: 25
Bob's age: 30
Charlie's age: 35
David's age: null
Bob's age after deletion: null
Size of hash table: 2
```