

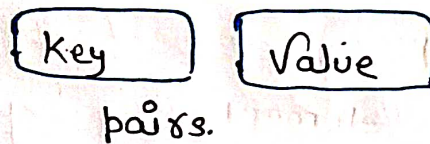
# HASH TABLE

Speciality:  $O(1)$  time complexity, for all the dictionary operations.

Hash table data structure stores elements in key-value pairs where

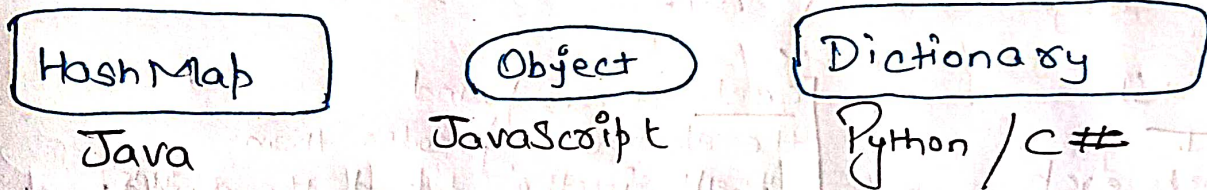
Key - Unique integers that is used for indexing the values.

Value - Data that are associated with keys.



Dictionary operations: INSERT, SEARCH AND DELETE

Different Names:



- In HashMaps, we can't have duplicate keys.

Methods:  $\text{Map} \left( \begin{matrix} \text{Data Type} \\ \text{Key} \end{matrix}, \begin{matrix} \text{Data Type} \\ \text{Value} \end{matrix} \right) \text{ map} = \text{new HashMap}();$

- `map.put()` // To add a key, value pair
- `map.remove()` // To remove
- `map.get(Key)` // Get the value at a particular key
- `map.containsKey()` // Return a boolean if the key is in the Hash Table.
- `map.containsValue()` // It has to go over all the elements, to check whether the value exists or not.  
 $\downarrow$   
 $O(n)$



v. map.keySet() // Returns all the keys  
vi. map.entrySet() // Returns all the key value pairs.

Maps: Key → Value

Sets: Key

- A set only contains keys. And it doesn't allow duplicates.

eg: Set <Integer> set = new HashSet <>();  
int [] numbers = {1, 2, 3, 3, 2, 1, 4};

```
for (int num : numbers)  
    set.add(num);
```

System.out.println(set) // output: {1, 2, 3, 4}

Methods:

set.add()

set.containsAll()

set.remove()

set.size()

set.removeAll()

set.clear()

set.contains()



## Hashing (Hash Function)

- In a hash table, a new index is processed using the keys. And the element corresponding to that key is stored in the index. This process is known as Hashing.

## Hash Collision

- When the hash function generates the same index for multiple keys, there will be a conflict (what value to be stored in that index). This is known as Hash Collision.

Simply: When two keys generate same index to store their values by going through some kind of hash function.

Resolving Hash Collisions:  $[\text{hash}_1 = \text{Key} \% \text{table-size}]$

In Integer form

→ Closed / Separate Chaining: Elements are stored in the same index by using a doubly linked list.

→ Open Addressing: By Probing (searching for an empty index).

(squared) → Linear Probing:  $(\text{hash}_1 + i) \% \text{table-size}$   
→ Quadratic Probing:  $(\text{hash}_1 + i^2) \% \text{table-size}$   
→ Double Hash:  $(\text{hash}_1 + i * \text{hash}_2) \% \text{table-size}$   
 $\text{hash}_2(\text{Key}): \text{prime} - (\text{Key} \% \text{prime})$

## Some basic algorithms:

To find the frequency of all the keys in Hash Map.

eg: str = " a mango man "

```
Map<String, Integer> map = new HashMap<>();
```

```
for (char Key: str.toCharArray())  
{
```

```
    int count = map.containsKey(Key) ? map.get(Key) : 0;  
    map.put(Key, count + 1);  
}
```

Count the frequency

It will grab the value at Key.

```
System.out.println(map);
```

```
// Output: {a=3, = 2, m=2, n=2, g=1, o=1}
```