HASHING

Java **HashSet class** is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface. HashSet contains unique elements only. HashSet allows **null value.**

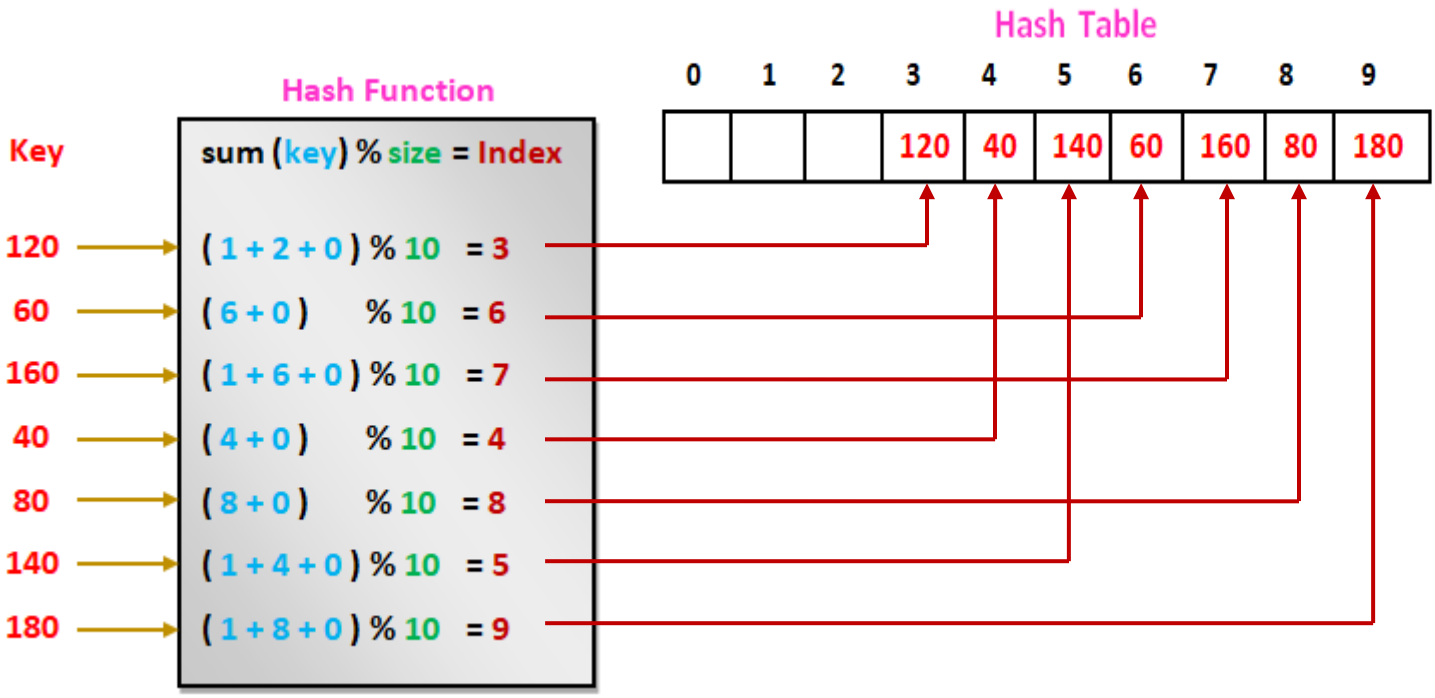
HashSet doesn't maintain any order, the elements would be returned in any random order. Here, elements are inserted on the basis of their hashcode. HashSet is the best approach for search operations.

HashSet class in java makes use of the ***hashing algorithm*** internally to store the data. Hashing makes use of a ***hash function*** and a ***hashtabe*** to store the data.

The duty of ***hashfunction*** is to take the data as input and calculate the location on the ***hashtable*** where the data must be stored.

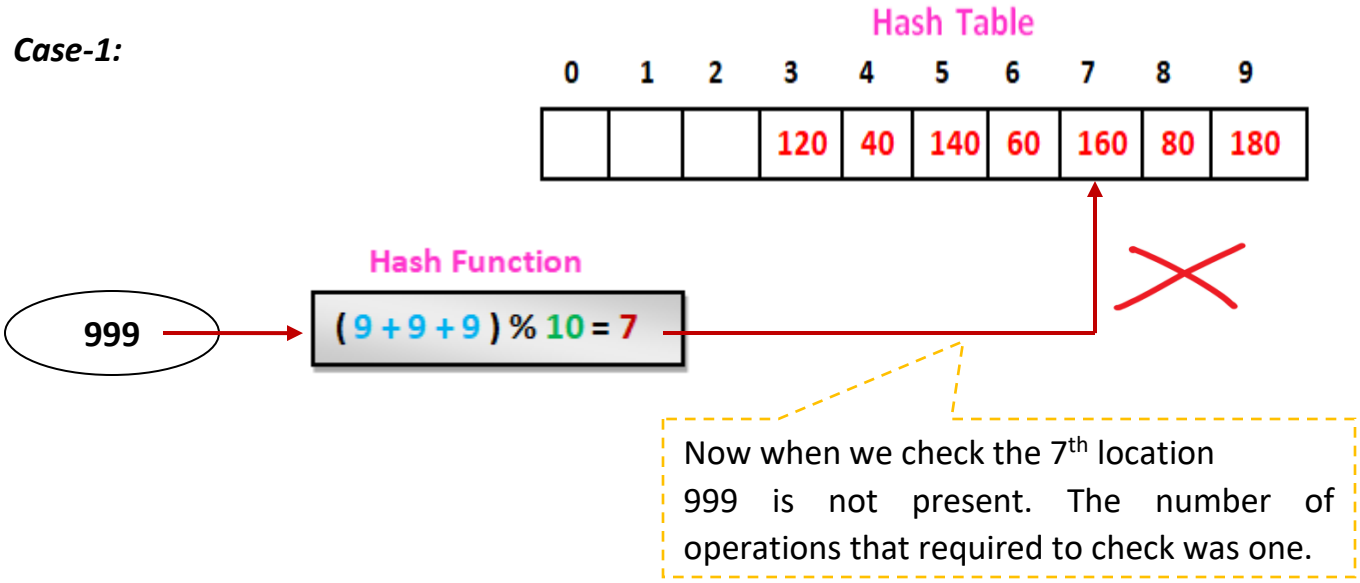
Let's create a hash function, such that our hash table has 10 numbers of buckets. To insert a code into the hash table, we need to find the ***hash index*** for the given key. And it could be calculated using the hash function.

hashIndex = sum(key) % size



Imagine we take some random value to check whether it is present or not in hash table, we make use of hash function with the formula.

Case-1:

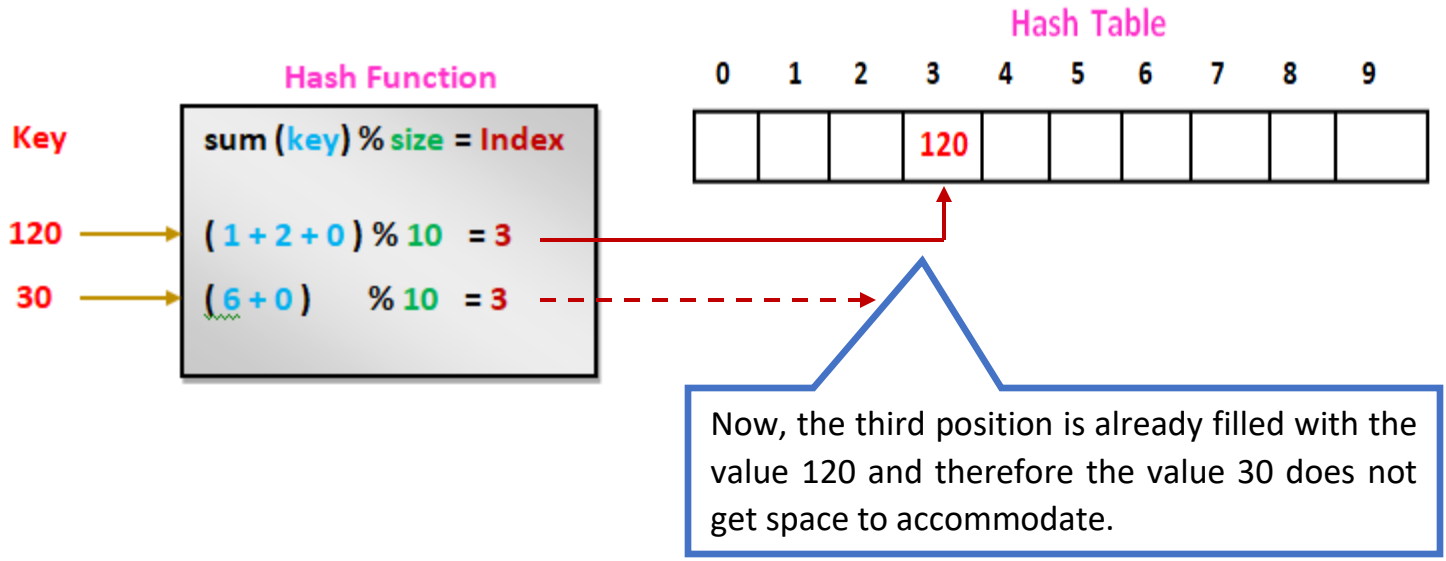


Number of Operation = 1

Hashing is the most efficient algorithm for performing search with a **time complexity of O(1).**

Let's look at the next Case-2:

Let us imagine we have two data which we pass it to the hash function to find its position in the hash table.



Sometimes two different pieces of data may generate at the same index given to the hash function and this is referred to as ***collision***.

To prevent collision **"Collision resolution technique"** is used such as making used of linked list at a point of collision etc.

Further the moment the capacity of the hashtable is filled to 75% its size is automatically doubled to reduce the chances of collision.

Code:

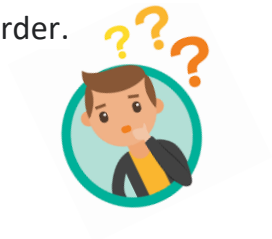
```
import java.util.HashSet;
class Demo
{
    public static void main(String[] args)
    {
        HashSet hs = new HashSet();
        hs.add(120);
        hs.add(60);
        hs.add(160);
        hs.add(40);
        hs.add(80);
        hs.add(140);
        hs.add(180);
        System.out.println(hs);
    }
}
```

Output

[160, 80, 180, 120, 40, 60, 140]

Press any key to continue…

As we see the output we can clearly understand that HashSet doesn't maintain any order. The elements would be returned in any random order.



But, how to maintain the order??

Well, for that we make use of LinkedHashSet.

The LinkedHashSet class extends HashSet class which implements Set interface. Java LinkedHashSet class contains unique elements only like HashSet. Java LinkedHashSet class provides all optional set operation and permits null elements.
LinkedHashSet class in Java makes use of the hashing algorithm internally to store the data.
Java LinkedHashSet class also preserves the order of insertion.

Code:

```
import java.util.LinkedHashSet;
class Demo
{
    public static void main(String[] args)
    {
        LinkedHashSet hs = new LinkedHashSet();
        hs.add(120);
        hs.add(60);
        hs.add(160);
        hs.add(40);
        hs.add(80);
        hs.add(140);
        hs.add(180);
        System.out.println(hs);
    }
}
```

Output:

[120, 60, 160, 40, 80, 140, 180]

Press any key to continue…

Differences and Similarities between ***HashSet*** *and* ***LinkedHashSet.***

HashSet	LinkedHashSet
1) HashSet internally uses HashMap for storing objects.	LinkedHashSet uses LinkedHashMap internally to store its elements.
2) HashSet doesn't maintain any order of elements.	In LinkedHashSet elements are placed as they are inserted.
3) HashSet gives performance of order O(1)	LinkedHashSet gives performance of order O(1)
4) HashSet requires less memory than LinkedHashSet as it uses only HashMap internally to store elements.	LinkedHashSet requires more memory than HashSet as it maintains LinkedList along with HashMap internally to store elements.
5) Use HashSet if you don't want to maintain any order of elements.	Use LinkedHashSet if you want to maintain insertion order of elements.