# CSCI 2120:
# Software Design & Development II

*UNIT 2: Collections Framework & Generics*

**Internal Workings of HashMap**

## Overview

## Introduction

In this lecture, we will learn how HashMap works in Java internally step-by-step with examples.

If you are not familiar with the basic features of Java HashMap, first go those lectures and then come back to learn the internal workings of HashMap in Java.

# HashMap in Java

**HashMap in Java** is basically an array of buckets (also known as bucket table of HashMap) where each bucket uses linked list to hold elements. A linked list is a list of nodes where each node contains a key-value pair.

In simple words, a bucket is a linked list of nodes where each node is an object of class Node<K,V>. The key of the node is used to obtain the hash value and this hash value is used to find the bucket from Bucket Table.

HashMap works on the principle of hashing data structure or technique that uses an object's hashcode to place that object inside the map.

Hashing involves Bucket, Hash function (hashCode() method), and Hash value. It provides the best time complexity of O(1) for insertion and retrieval of objects.

Therefore, it is the best-suited data structure for storing key-value pairs that later on can be retrieved in minimum time.

# HashMap in Java

A typical hashing data structure for storing key-value pairs can be seen in the below figure.

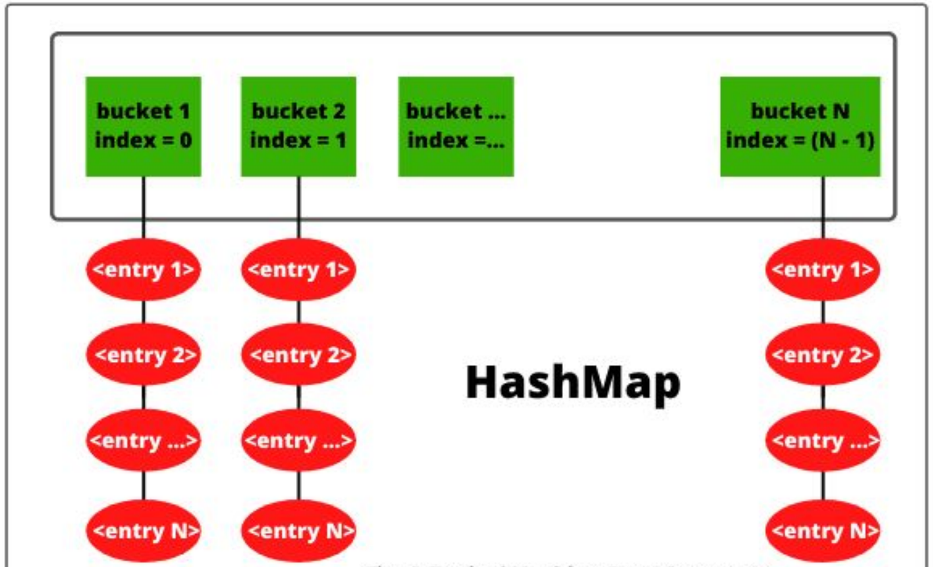

Fig: A Typical Hashing Data Structure

## Initial Capacity and Load Factor of Java HashMap

Load Factor and Initial Capacity are two important factors that plays important role in the internal working of HashMap in Java.

**Initial Capacity** is a measure of the number of buckets or size of bucket array internally by HashMap at the time of the creation of HashMap.

The default initial capacity of HashMap is 16 (i.e. the number of buckets). It is always expressed in the power of 2 (2, 4, 8, 16, etc) reaching maximum of 1 << 30 (2^30).

**Load Factor** is a factor that is internally used by HashMap to determine when the size of Bucket array requires to be increased. By default, it is 0.75.

When the number of nodes in the HashMap is more than 75% of total capacity, HashMap grows its bucket array size. The capacity of HashMap always doubled each time when HashMap needs to be increased its bucket array size.

## Initial Capacity and Load Factor of Java HashMap

**Bucket Table:**

An array of buckets is called bucket table of HashMap. In the bucket table, a bucket is a linked-list of nodes where each node is an object of class Node<K, V>.

The key of node is used to obtain the hash value and this hash value is used to calculate the index of the bucket from the bucket table in which key-value pairs will be placed.

## Initial Capacity and Load Factor of Java HashMap

**Node:**

Each node of the linked-list is an object of class `Node<K,V>`. This class is a static inner class of `HashMap` class that implements the `Map.Entry<K,V>` interface.

The general syntax for inner static `Node` class of `HashMap` is as follows:

```
static class Node<K,V> implements Map.Entry<K,V> {
   final int hash;
   final K key;
   V value;
   Node<K,V> next;
}
```

| | |
|---|---|
| `final int hash:` | It is a hash value of the key. |
| `final K key:` | It is a key of the node. |
| `V value:` | It gives the value of the node. |
| `Node<K,V> next:` | It indicates the pointer to the next node present in the bucket or linked-list. |

## Role of hashCode() & equals() in Internals of HashMap

The hashcode() and equals() play a major role in the internal working of HashMap in Java. Therefore, before understanding the internal working of HashMap, you should be aware of basic knowledge of hashCode() and equals() methods.

## Role of hashCode() & equals() in Internals of HashMap

**hashCode():**

Hash function is a function that maps a key to an index in the hash table. It obtains an index from a key and uses that index to retrieve the value for a key.

A hash function first converts a search key (object) to an integer value (known as hash code) and then compresses the hash code into an index to the hash table.

The Object class (root class) of Java provides a hashCode method that other classes need to override. hashCode() method is used to retrieve the hash code of an object. It returns an integer hash code by default that is a memory address (memory reference) of an object.

## Role of hashCode() & equals() in Internals of HashMap

**hashCode():**

The general syntax for the hashCode() method is as follows:

```
public native hashCode()

//The general syntax to call hashCode() method is as follows:
int h = key.hashCode();
```

The value obtained from the hashCode() method is used as the bucket index number. The bucket index number is the address of the entry (element) inside the map. If the key is null then the hash value returned by the hashCode() will be 0.

## Role of hashCode() & equals() in Internals of HashMap

**equals():**

The `equals()` method is a method of Object class that is used to check the equality of two objects. HashMap uses `equals()` method to compares Keys whether they are equal or not.

The equals() method of Object class can be overridden. If we override the `equals()` method, it is mandatory to override the `hashCode()` method.

# Put Operation

**How put() method of Hashmap works internally in Java?**

The put() method of HashMap is used to store the key-value pairs. The syntax of put() method to add key/value pair is as follows:

```
hashmap.put(key, value);
```

Let’s take an example where we will insert three (Key, Value) pairs in the HashMap.

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

hmap.put("John", 20);
hmap.put("Harry", 5);
hmap.put("Deep", 10);
```

## Put Operation

**How put() method of Hashmap works internally in Java?**

Let’s understand at which index the key-value pairs will be stored into HashMap.

When we call the put() method to add a key-value pair to hashmap, HashMap calculates a hash value or hash code of key by calling its hashCode() method. HashMap uses that code to calculate the bucket index in which key/value pair will be placed.

The formula for calculating the index of bucket (where n is the size of an array of the bucket) is given below:

```
Index = hashCode(key) & (n-1);
```

## Put Operation

**How put() method of Hashmap works internally in Java?**

Suppose the hash code value for “John” is 2657860. Then the index value for “John” is:

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

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

**Note:** Since HashMap allows only one null Key, the hash value returned by the hashCode(key) method will be 0 because the hashcode for null is always 0. The 0th bucket location will be used to place key/value pair.

## How is Hash Collision occurred and resolved?

A hash collision occurs when hashCode() method generates the same index value for two or more keys in the hash table. To overcome this issue, HashMap uses the technique of linked-list.

When hashCode() method produces the same index value for a new Key and the Key that already exists in the hash table, HashMap uses the same bucket index that already contains nodes in the form of linked-list.

A new node is created at the last of the linked-list and connect this node object to the existing node object through the LinkedList. Hence both Keys will be stored at the same index value.

When a new value object is inserted with an existing Key, HashMap replaces the old value with the current value related to the Key. To do it, HashMap uses equals() method.

This method check that both Keys are equal or not. If Keys are the same, this method returns true and the value of that node is replaced with the current value.

## How get() method in HashMap works internally in Java?

The get() method in HashMap is used to retrieve the value by its key. If we don’t know the Key, it will not fetch the value. The syntax for calling get() method is as follows:

```
value = hashmap.get(key);
```

When the get(K Key) method takes a Key, it calculates the index of bucket using the method mentioned above. Then that bucket’s List is searched for the given key using equals() method and final result is returned.

## Time Complexity of put() and get() methods

HashMap stores a key-value pair in constant time which is O(1) for insertion and retrieval. But in the worst case, it can be O(n) when all node returns the same hash value and inserted into the same bucket.

The traversal cost of n nodes will be O(n) but after the changes made by Java 1.8 version, it can be maximum of O(log n).

## Concept of Rehashing

**Rehashing** is a process that occurs automatically by HashMap when the number of keys in the map reaches the threshold value. The threshold value is calculated as **threshold** = capacity * (load factor of 0.75).

In this case, a new size of bucket array is created with more capacity and all the existing contents are copied over to it.

For example:

```
Load Factor: 0.75
Initial Capacity: 16 (Available Capacity initially)
Threshold = Load Factor * Available Capacity = 0.75 * 16 = 12
```

## Concept of Rehashing

When 13th key-value pair is inserted into the HashMap,
HashMap grows its bucket array size to 16*2 = 32.

```
Now Available capacity: 32

Threshold = Load Factor * Available Capacity = 0.75 * 32 = 24
```

Next time when 25th key-value pair is inserted into HashMap, HashMap grows its bucket array size to 32*2 = 64 and so on.

# END