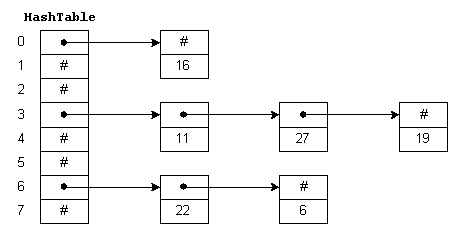
**Java Hash table class**

**Java Hash table** class is an implementation of hash table data structure. It is very much similar to [HashMap](https://howtodoinjava.com/java-hashmap/) in Java, with most significant difference that Hash table is **synchronized** while HashMap is not.

## How Hash table Works?

Hash table internally contains buckets in which it stores the key/value pairs. The Hash table uses the key’s [hash code](https://howtodoinjava.com/java/basics/java-hashcode-equals-methods/) to determine to which bucket the key/value pair should map.



The function to get bucket location from Key’s hash code is called [**hash function**](https://howtodoinjava.com/java/java-security/how-to-generate-secure-password-hash-md5-sha-pbkdf2-bcrypt-examples/). In theory, a hash function is a function which when given a key, generates an address in the table. A hash function always returns a number for an object. Two equal objects will always have the same number while two unequal objects might not always have different numbers.

When we put objects into a hash table, it is possible that different objects (by the equals () method) might have the same hash code. This is called a **collision**. To resolve collisions, hash table uses an [**array**](https://howtodoinjava.com/java-array/)**of lists**. The pairs mapped to a single bucket (array index) are stored in a list and list reference is stored in array index.

## 2. Hashtable Features

The important things to learn about Java Hashtable class are:

1. It is similar to HashMap, but it is synchronized while HashMap is not [synchronized](https://howtodoinjava.com/java/multi-threading/what-is-thread-safety/).
2. It does not accept null key or value.
3. It does not accept duplicate keys.
4. It stores key-value pairs in hash table data structure which internally maintains an array of list. Each list may be referred as a bucket. In case of collisions, pairs are stored in this list.

## 3. Hashtable Constructors

Hashtable class has four constructors.

* **Hashtable():** It is the default constructor. It constructs a new, empty hashtable with a default initial capacity (11) and load factor (0.75).
* **Hashtable(int size):** It constructs a new, empty hashtable of specified initial size.
* **Hashtable(int size, float fillRatio):** It constructs a new, empty hashtable of specified initial size and fill ratio.
* **Hashtable(Map m):** It constructs a hashtable that is initialized with the key-value pairs in specified map.

Please note that **initial capacity** refers to number of buckets in hashtable

Underlining data structure for HashTable is HashTable

Duplicate Key not allowed but duplicate values are allowed

Null and value are not allowed

HashTable not allowed random access

Synchronized so it is thread safe

Best choice for search operation

Default initial capacity of a hashtable object is 11.and load factor is 0.75

**public** **class** HashTable1

{

**public** **static** **void** main(String[] args)

{

Hashtable<Integer, String> ht =**new** Hashtable<Integer, String>();

ht.put(10, "Golaghat");

ht.put(20, "Jorhat");

ht.put(30, "Tezpur");

ht.put(5, "Lakhimpur");

ht.put(12, "Jayanta");

**for**(Map.Entry<Integer, String> entry : ht.entrySet())

{

System.***out***.println(entry.getKey() + ":"+ entry.getValue());

}

}

}

Since initial capacity is 11

In a bucket we can place multiple entry

|  |  |
| --- | --- |
| Array of List | Buckets |
| 0 |  |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 | 15=Jayanta |
| 5 | 5=Lakhimpur, 16=Lahkar |
| 6 | 6=Tezpur |
| 7 |  |
| 8 |  |
| 9 | 20=Jorhat |
| 10 | 10=Golaghat |

ht.put(10, "Golaghat");

ht.put(20, "Jorhat"); 20%11=9

ht.put(6, "Tezpur");

ht.put(5, "Lakhimpur");

ht.put(15, "Jayanta"); 15%11=4

ht.put(16, "Lahkar"); 16%11=5

From the Bucket output will be generated from top to bottom and right to left

Output is

10:Golaghat

20:Jorhat

6:Tezpur

16:Lahkar

5:Lakhimpur

15:Jayanta

Hashtable<Integer, String> ht =**new** Hashtable<Integer, String>(20);

Here we change the initial capacity of the hashtable to 20

And output will be changed because index of the bucket are changed. Key%size=reminder

16:Lahkar

15:Jayanta

10:Golaghat

6:Tezpur

5:Lakhimpur

20:Jorhat