The **Hashtable** class implements a hash table, which maps keys to values. It inherits the Dictionary class and implements the Map interface. Any non-null object can be used as a key or as a value. To successfully store and retrieve objects from a hashtable, the objects used as keys must implement the hashCode method and the equals method.

Features:

* It is similar to HashMap, but is synchronized.
* Hashtable stores key/value pair in hash table.
* In Hashtable, er specify an object that is used as a key, and the value we want to associate to that key. The key is then hashed, and the resulting hash code is used as the index at which the value is stores within the table.
* The initial default capacity of Hashtable class is 11 whereas loadFactor is 0.75.
* HashMap doesn’t provide any Enumeration, while Hashtable provides not fail-fast Enumeration.

Points to Remember:

* A Hashtable is an array of a list. Each list is known as a bucket. The position of the bucket is identified by calling the hashcode() method. A Hashtable contains values based on the key.
* Java Hashtable class contains unique elements.
* It does not allow null key or value.
* It is synchronized.

Declaration

public class Hashtable<K, V> extends Dictionary<K, V> implements Map<K, V>, Cloneable, Serializable

K- the type of keys maintained by this map

K- the type of mapped values

In order to create a Hashtable, we need to import it from java.util.Hashtable. There are various ways in which we can create a Hashtable.

1. Hashtable(): this creates an empty hashtable with the default load factor of 0.75 and an initial capacity of 11.

Hashtable<K, V> ht = new Hashtable<K, V>();

1. Hashtable(int initialCapacity): this creates a hash table that has an initial size specified by initialCapacity and the default load factor is 0.75.

Hashtable<K, V> ht = new Hashtable<K, V>(int initialCapacity);

1. Hashtable(int size, float fillRatio): this version creates a hash table that has an initial size specified by size and fill ration specified by fillRatio. Fill ration basically determines how full a hash table can be before it is resized upward and its Value lies between 0.0 to 1.0.

Hashtable<K, V> ht = new Hashtable<K, V>(int size, float fillRatio);

1. Hashtable(Map<? Extends K, ? extends V> m): this creates a hash table that is initialized with the elements in m.

Hashtable<K, V> ht = new Hashtable<K, V>(Map m);

Performing Various Operations on Hashtable

1. Adding Elements

To add elements to the hash table, you will have to use the put() method. However, the insertion order is not retained in the hashtable. Internally, for every element, a separate hash is generated and the elements are indexed based on this hash to make it more efficient.

1. Changing Elements

After adding the elements, if we wish to change the element, it can be done by again adding the element with the put() method. Since the elements in the hashtable are indexed using the keys, the value of the key can be changed by simply inserting the updated value for the key for which we wish to change.

1. Removing Element

In order to remove an element from the Map, we can use the remove() method. This method takes the key value and removes the mapping for a key from this map if it is present in the map.

1. Traversal of a Hashtable

To iterate the table, we can make use of an advanced for-loop.