Internal working of HashMap –

- It is just key value store, so not added in Collection i.e. Map does not extend Collection interface.

<p>The <tt>Map</tt> interface provides three <i>collection views</i>, which

\* allow a map's contents to be viewed as a set of keys, collection of values,

\* or set of key-value mappings.

- implementation of Map interface – hierarchy -

HashMap<K,V> extends AbstractMap<K,V>

And

public abstract class AbstractMap<K,V> implements Map<K,V>

And

public interface Map<K,V> {

- Works on principle of hashing

- Stores values as Map.Entry<K, V>

- Uses int hashCode() – native method – for calculating the index/ bucket value for storing the key value pair

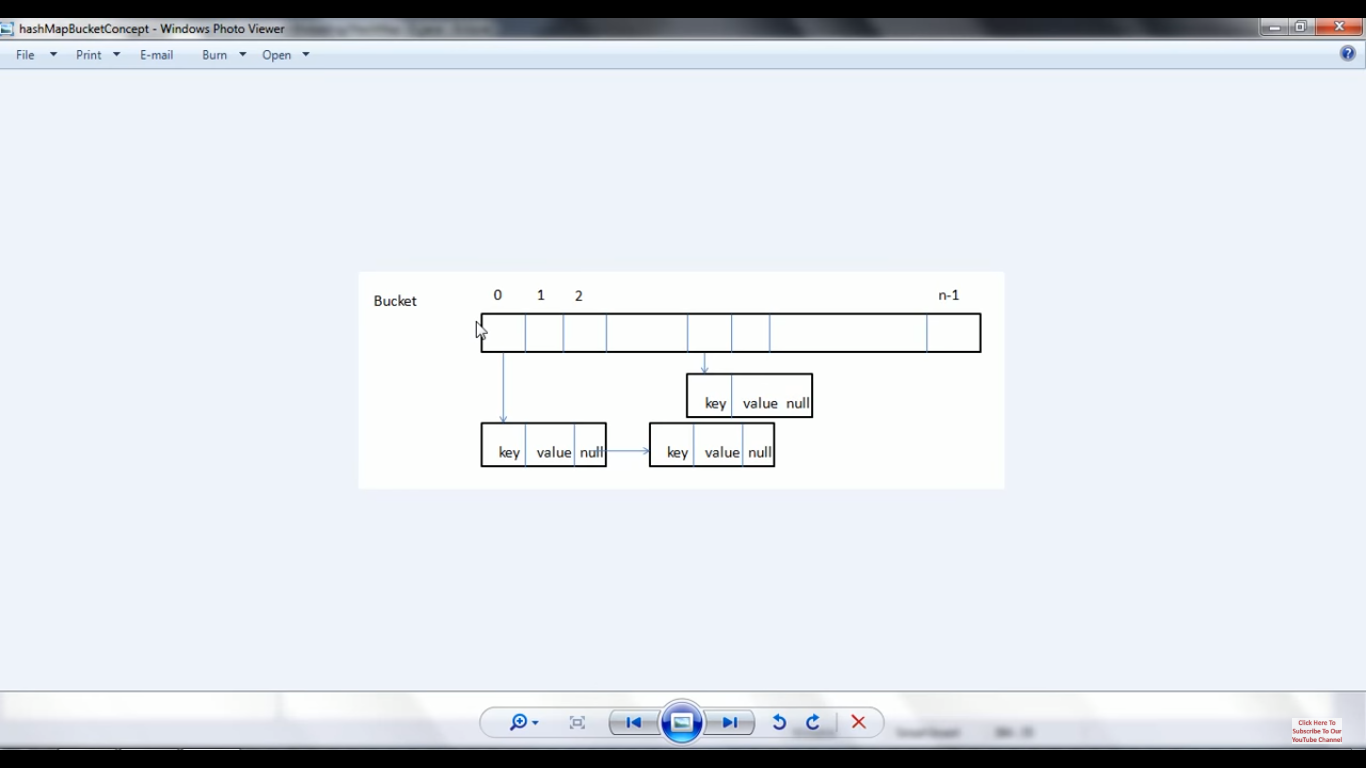
- equals()- Object class method - Rescues when 2 or more keys have same hashCode (bucket index) and when we are trying to get value (OR even while storing)

HashMap has initial capacity – default 16.

You can also specify by 1 parameter constructor.

Put operation –

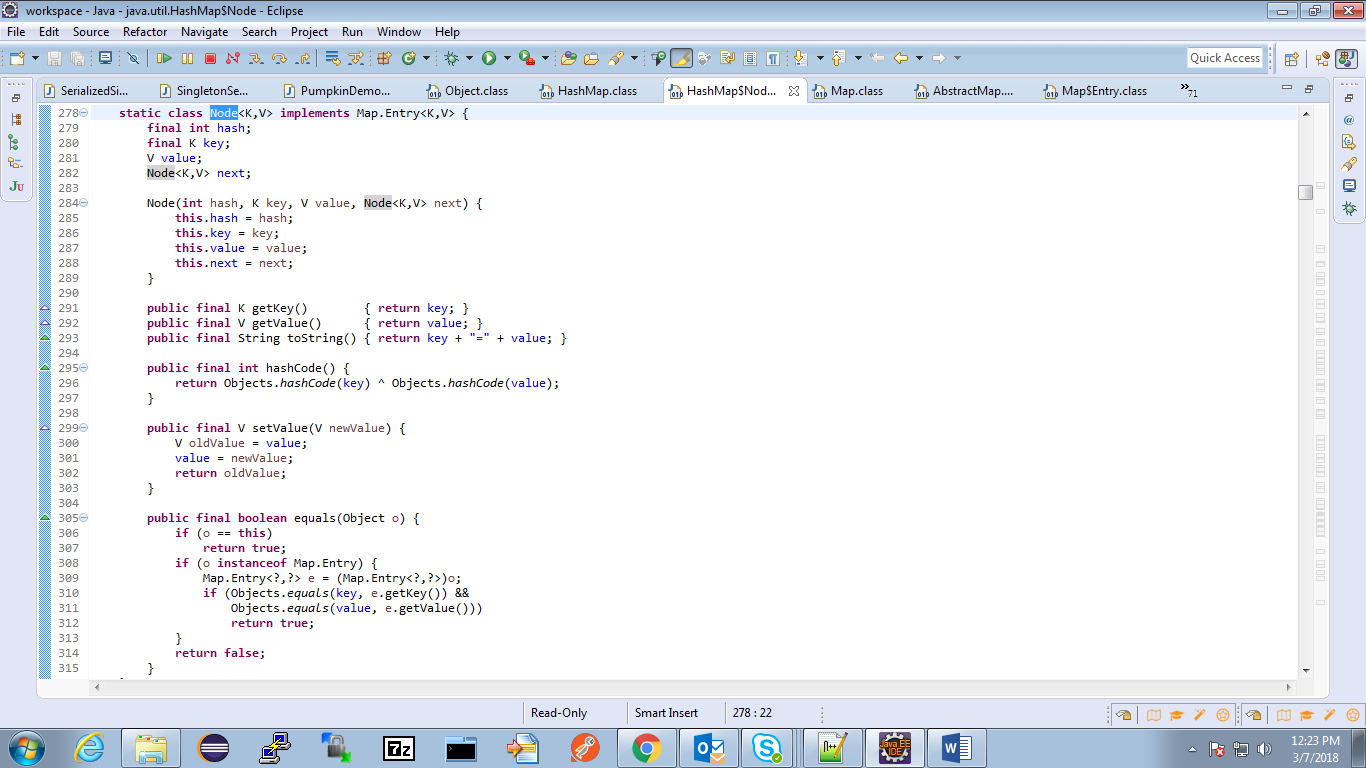
When we pass null as key – add the Entry in 0th bucket position (of an array that has each element as LinkedList)

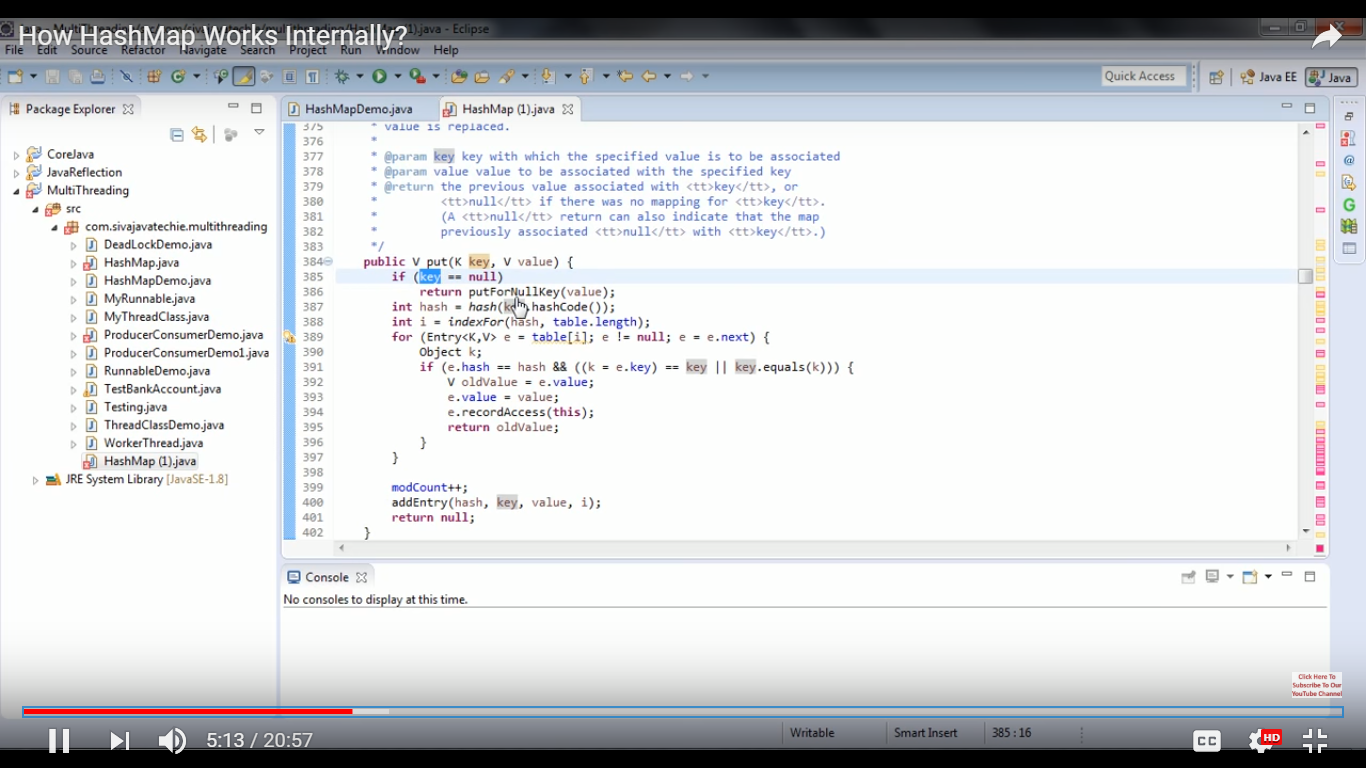


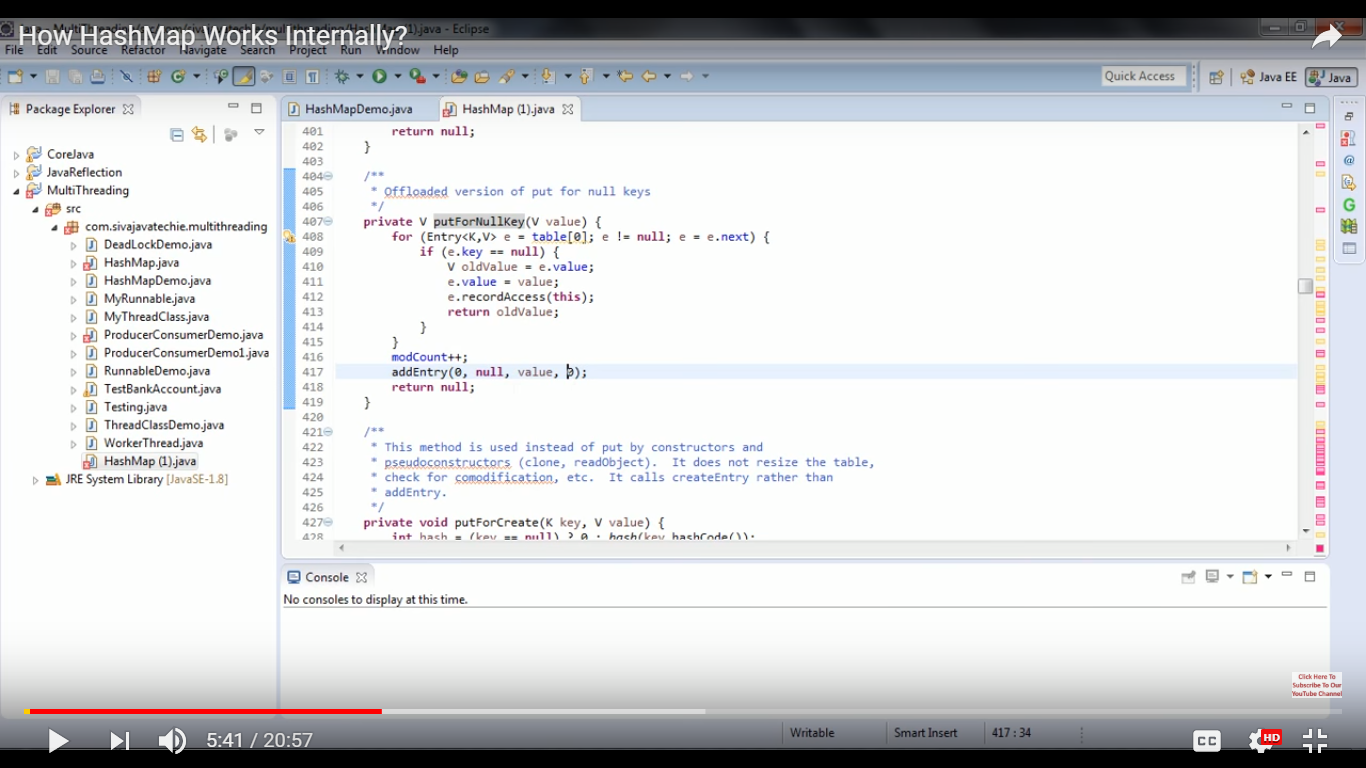
- table is an array of type Node<K, V>

transient Node<K,V>[] table;

And Node is a static inner class like –







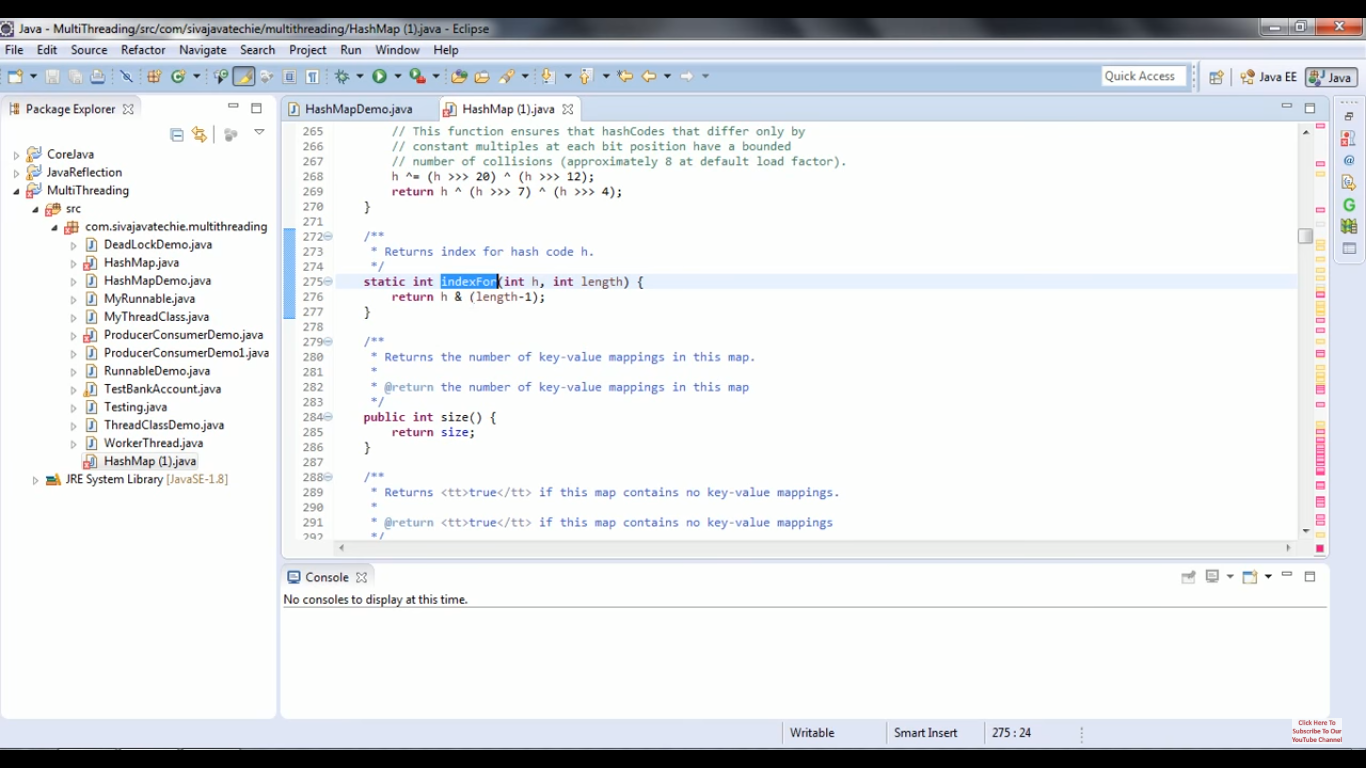
If key which is not null – For duplicate keys – previous key value pair is replaced – no any error

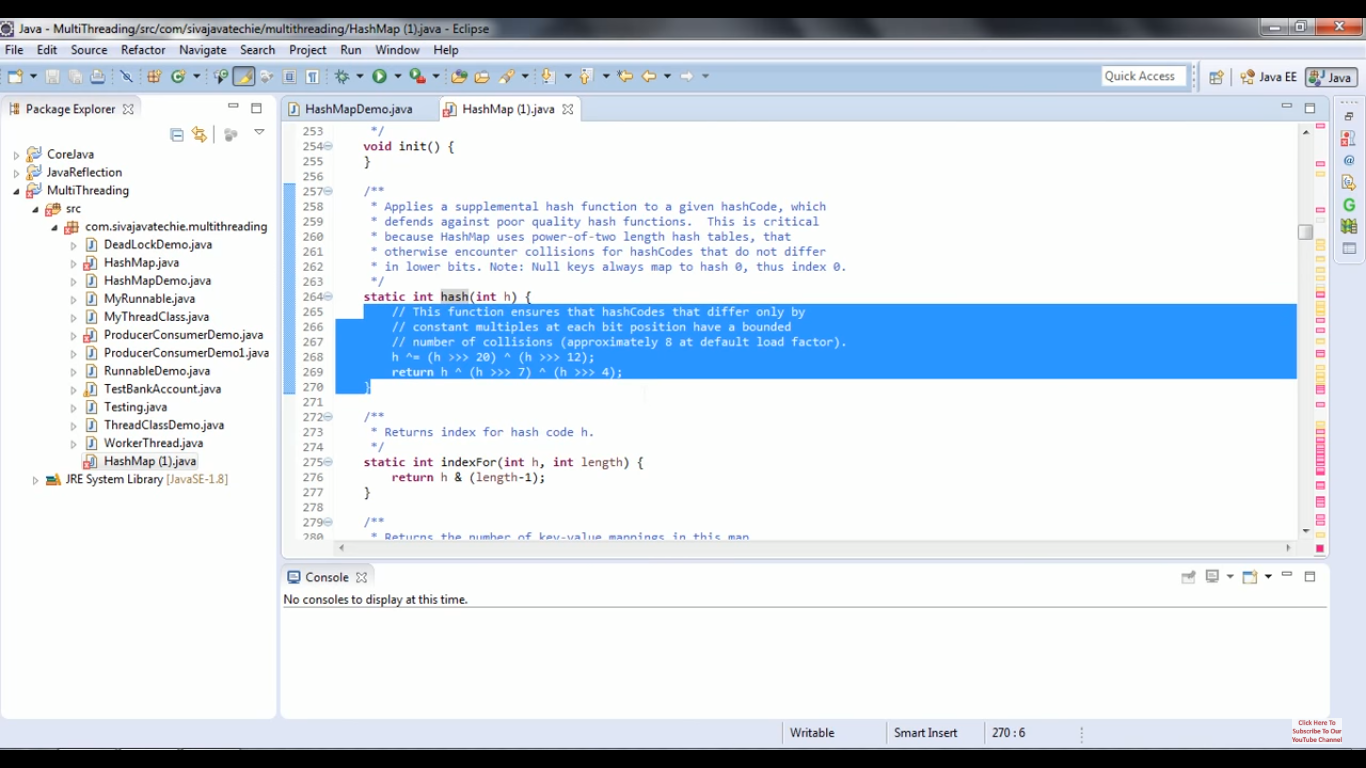
Calculate the 2 level hash to make sure that has values (in turn bucket locations) are not repeated

– first get the key.hashCode() and apply hash on it.

int hash = hash(key.hashCode());

- Use indexFor() – gives index for the hash passed to it





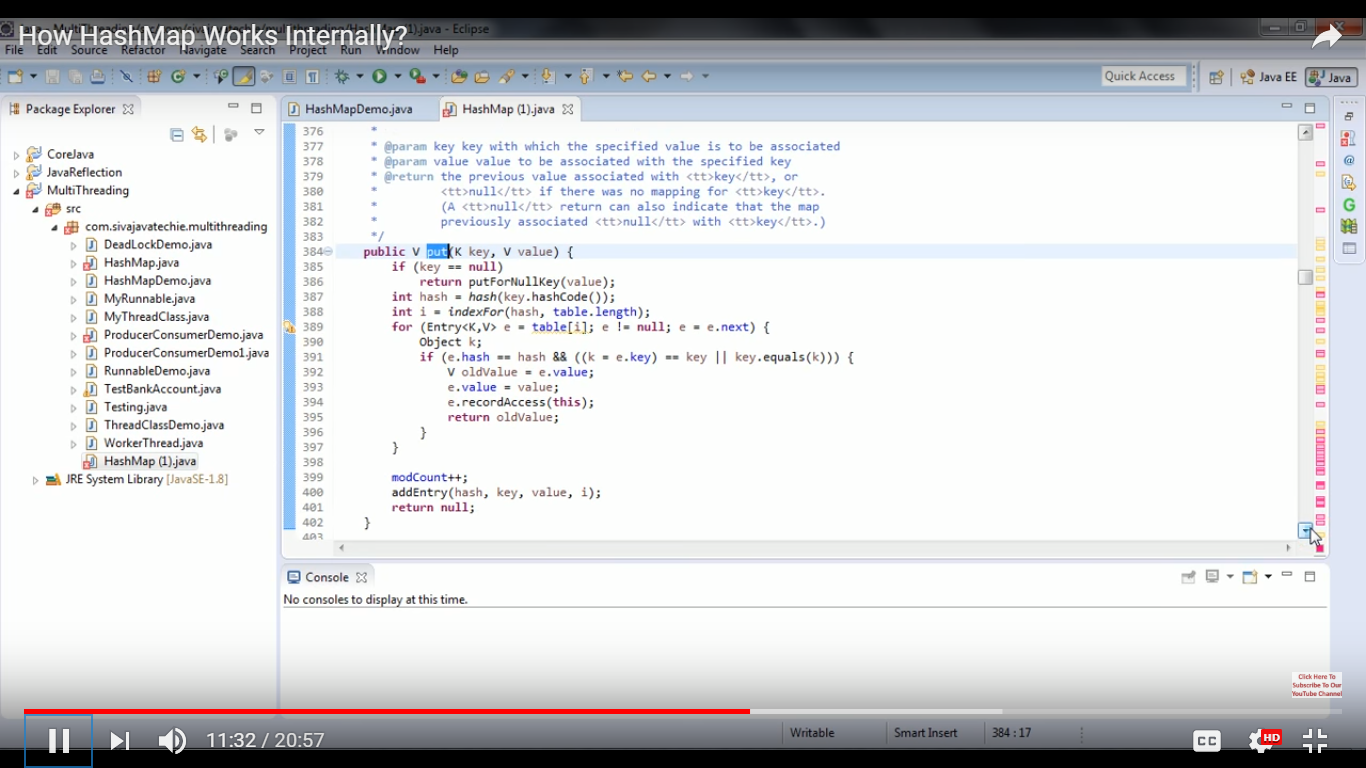
When indexFor() returns the array (named ‘table’) index location, an entry is made there. It will be check if any Entry (ies) already exists there. If yes, their key equality is checked.

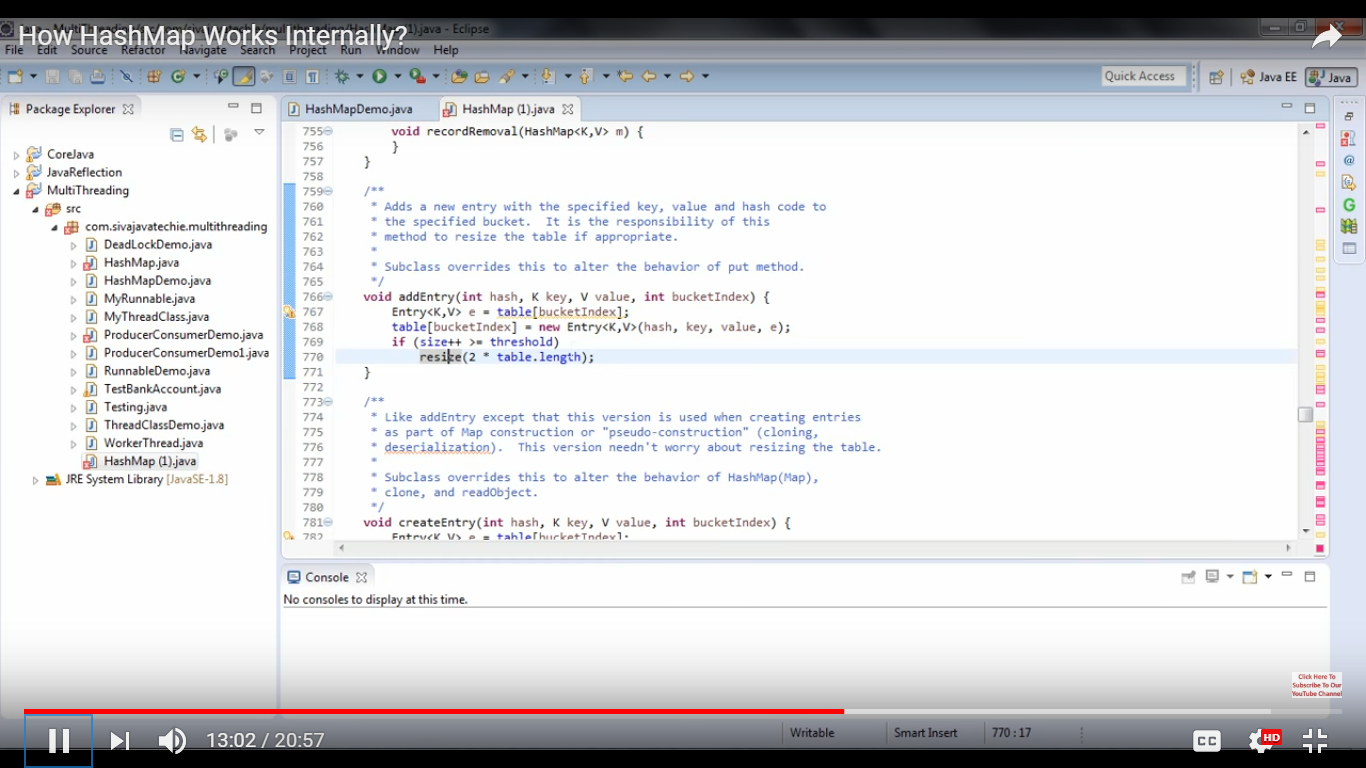
- If keys are same, then old Entry will be replaced.

- If keys are different OR no Entry exist at that table[i]; then a new Entry will be added

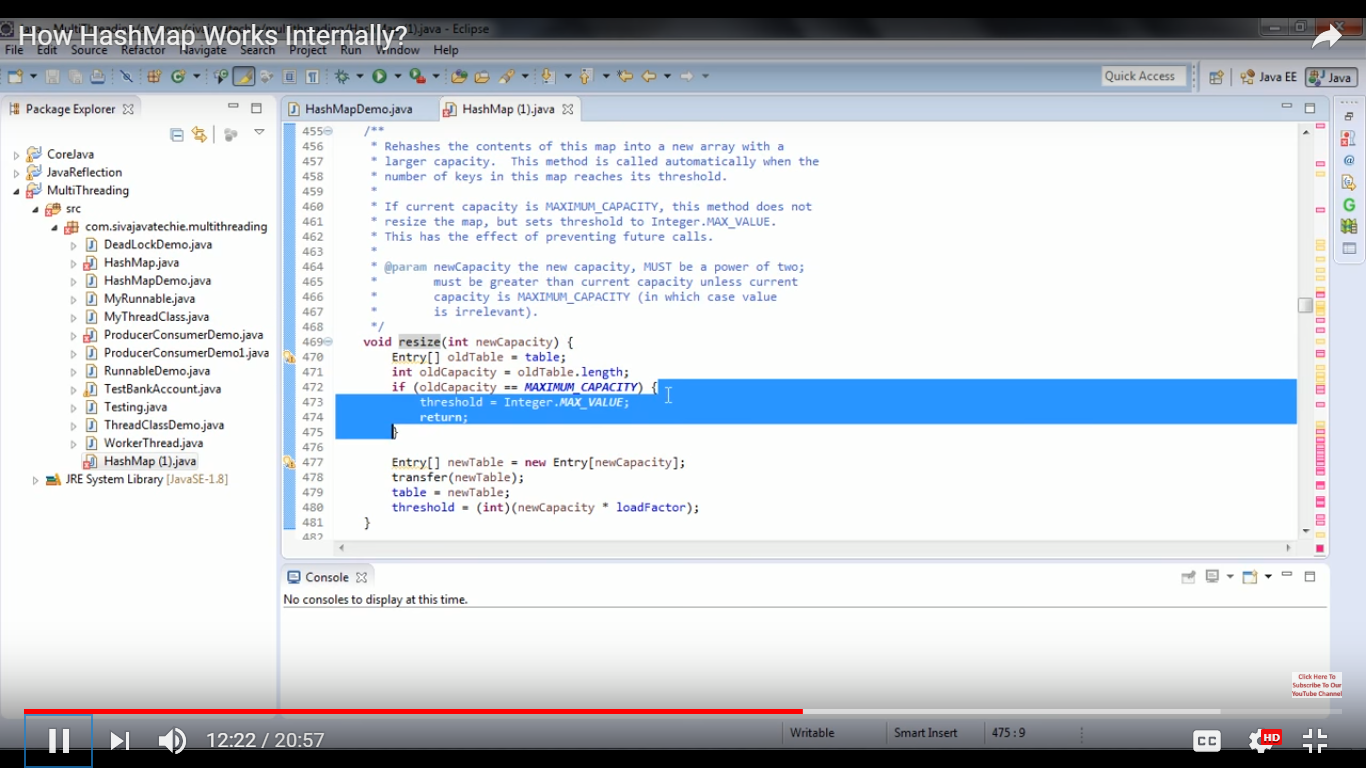
🡪 by calling addentry() that accepts 4 params – hash, key, value and index

If two different keys have same hash value and index returned is same; then a linked list is maintained at that array (table) location.

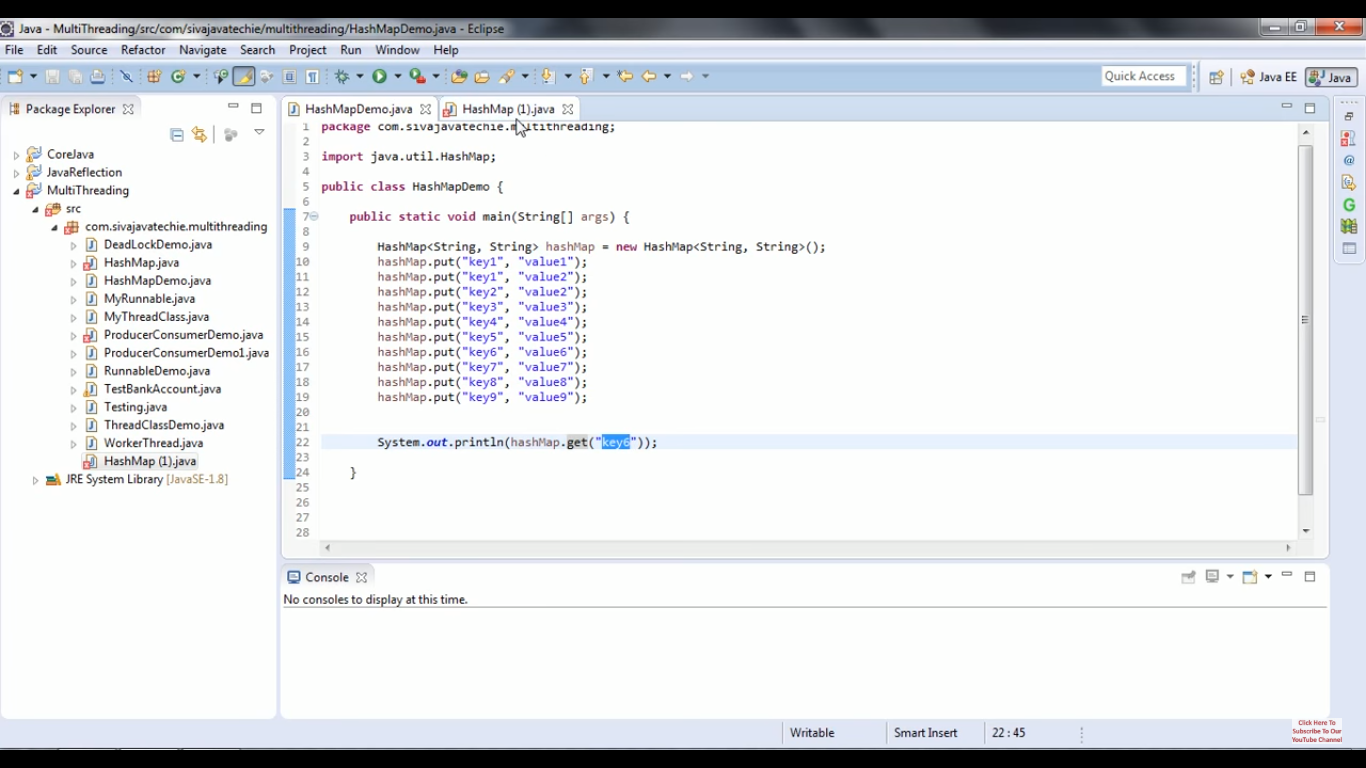




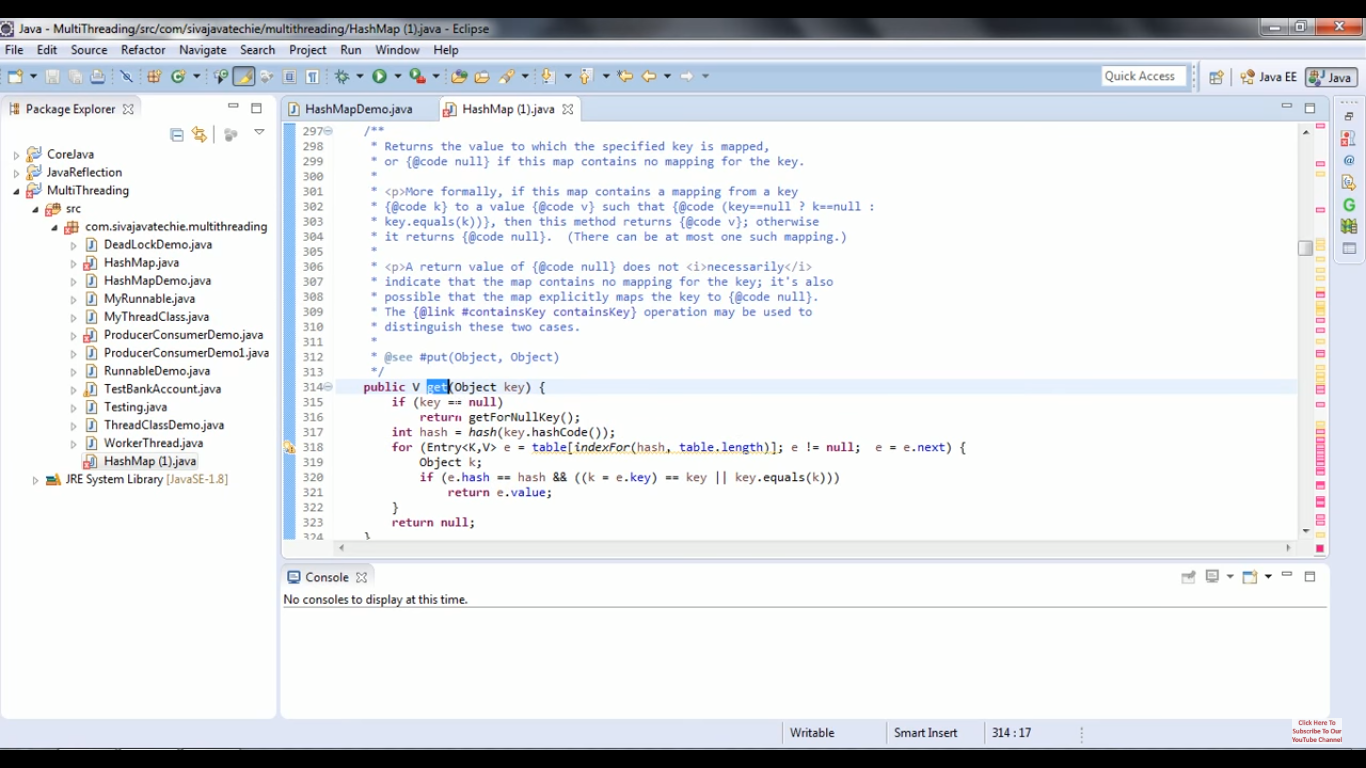
If size of the map reaches threshold, size is doubled -

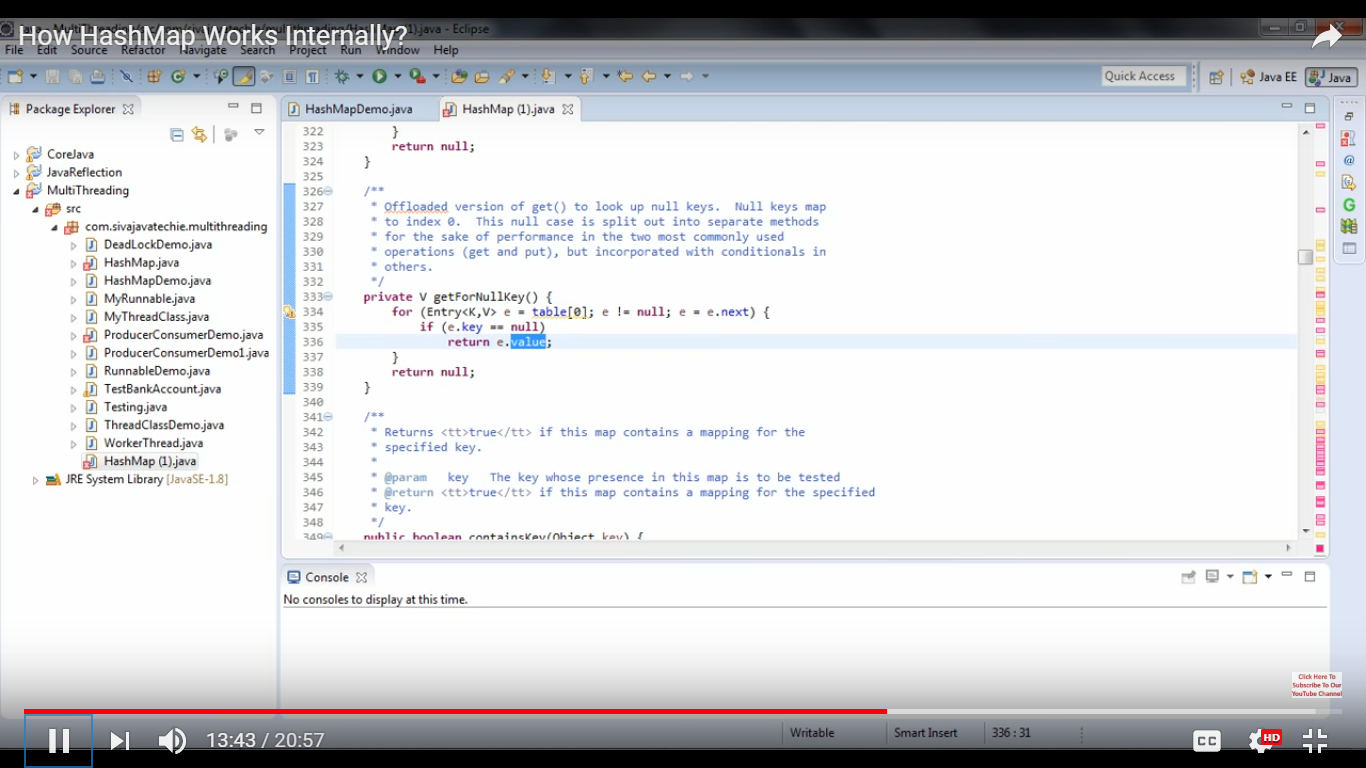


GET implementation –



Check if key is null – If yes, get value from 0th index -

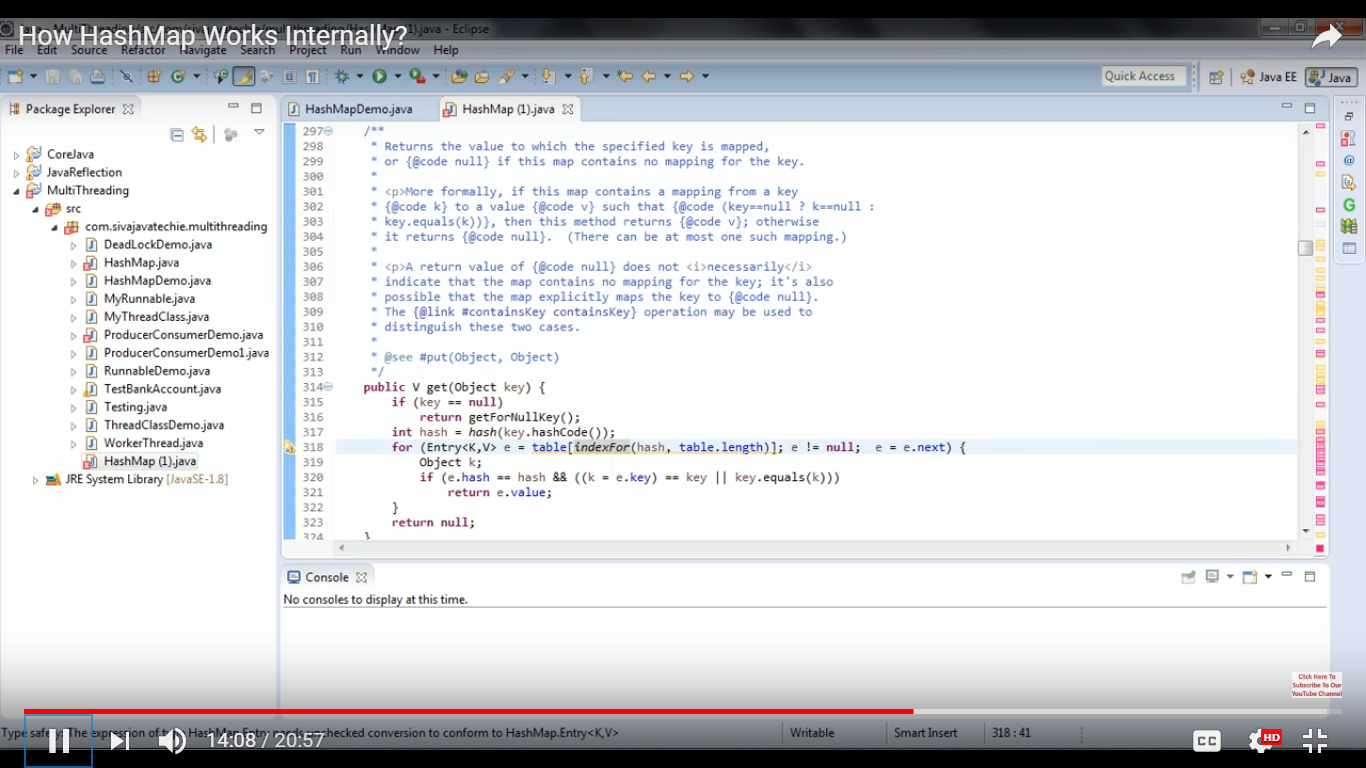


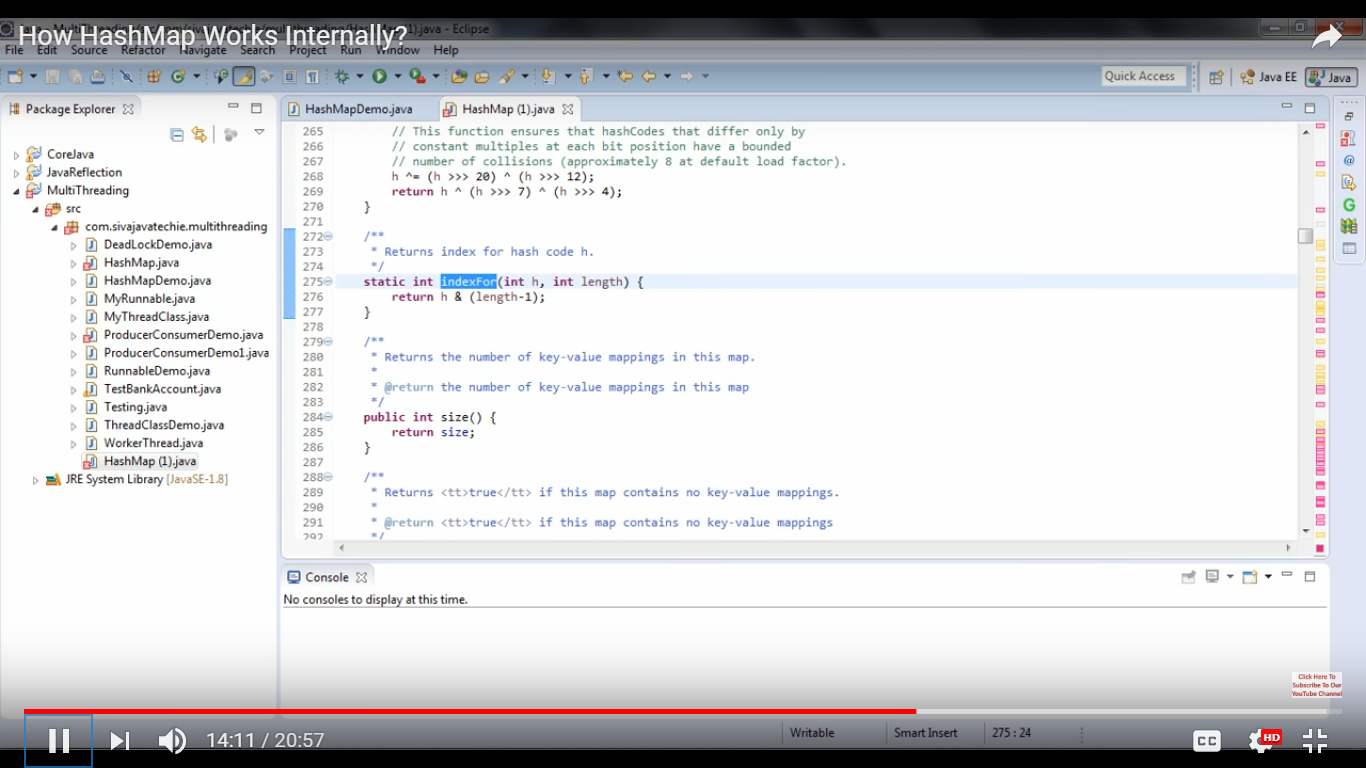


If not null key –

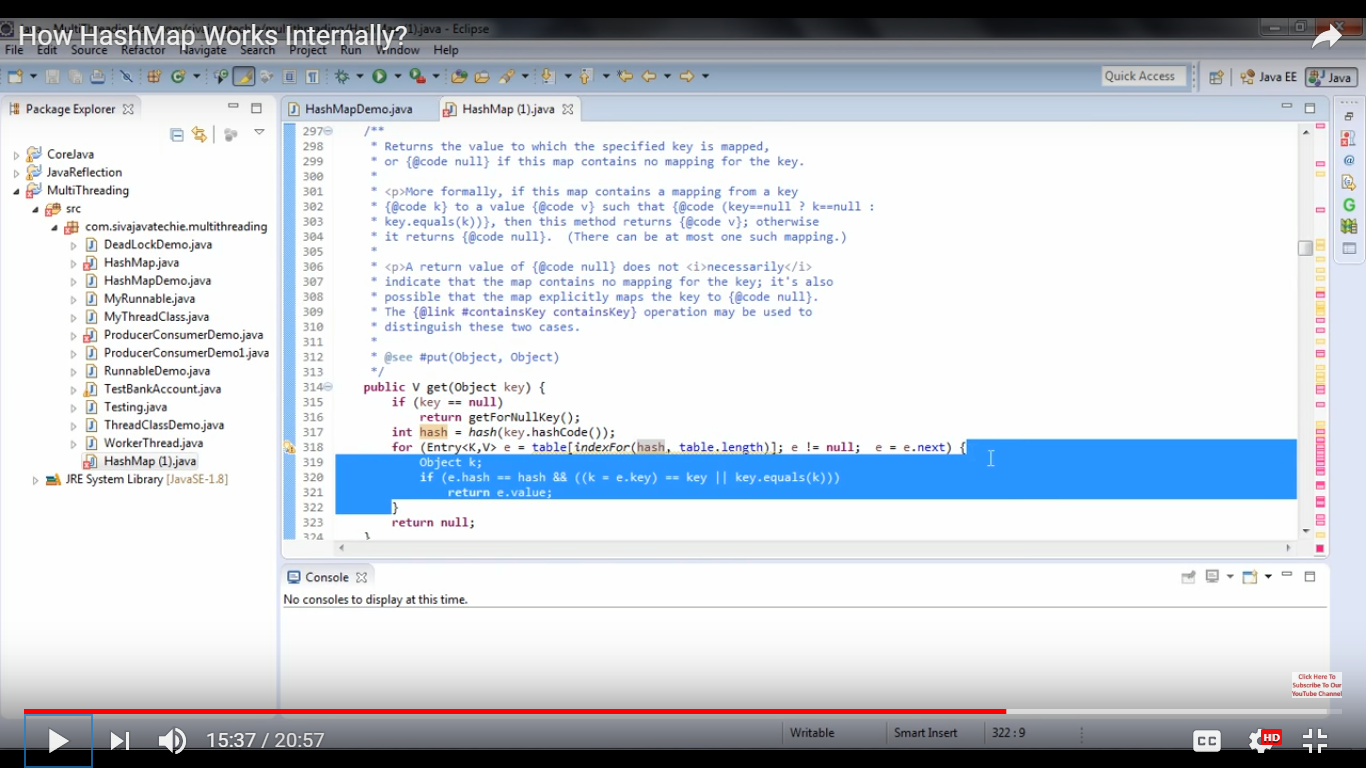
Calculate the hash value on the provided key using same 2 level hashing as done while putting the value.

Get the index for the hashed value and see if it has some Entry e and it’s not null.





Once index is found, return the value from that bucket/ index location by checking if the key at that index is same as what we have passed to get()



PERFORMANCE –

For get and put operations – as per javadocs

O(1) – if you have appropriate hash function i.e. Order of 1. Used for both storing and retrieving value.

- Load factor – 0.75 and initial capacity (default) – 16 (2 to the power of 4). Always increased in power of 2 (doubled). i.e. When 2^4 is filled 75% (12 indexes are occupied), then make it of size 2^5

- Capacity \* load factor => Threshold

- Collision – when 2 or more keys have same hash code

Note –

In java 8, it uses Balanced trees (TreeNode, an implementation of red-black tree links) instead of Linked list