* How HashSet works internally in java?

**HashSet** uses ***HashMap*** internally to store it’s objects. Whenever you create a HashSet object, one ***HashMap*** object associated with it is also created. This HashMap object is used to store the elements you enter in the HashSet. The elements you add into HashSet are stored as ***keys*** of this HashMap object. The value associated with those keys will be a ***constant***.

HashSet belongs to rt.jar.

**public** **class** **HashSet**<E>

**extends** AbstractSet<E>

**implements** Set<E>, Cloneable, java.io.Serializable

{

**private** **transient** HashMap<E,Object> map;

// Constant Dummy value to associate with an Object in the backing Map

**private** **static** **final** Object PRESENT = **new** Object();

**public** **HashSet**() {

*map =* ***new*** *HashMap<>();*

}

// SOME CODE ,i.e Other methods in Hash Set

**public** **boolean** **add**(E e) {

**return** map.put(e, PRESENT)==**null**;

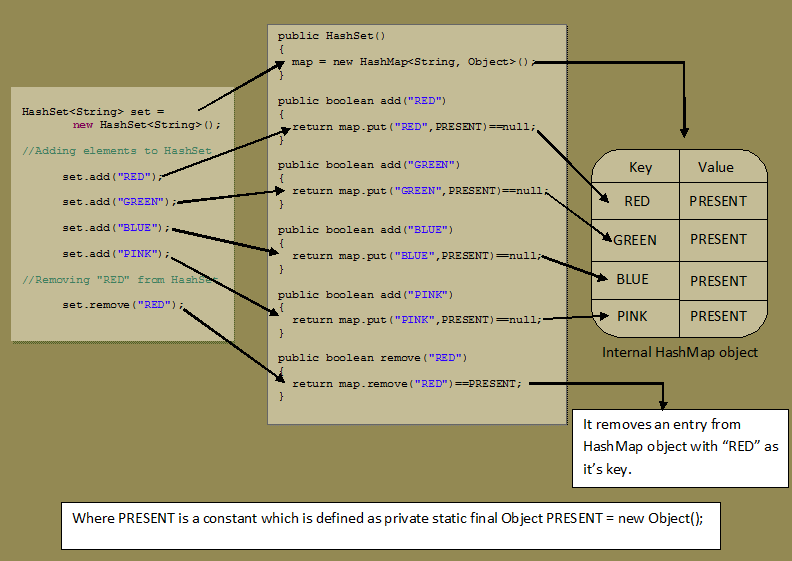
}

// SOME CODE ,i.e Other methods in Hash Set

}

The main point to notice in above code is that put (key,value) will return  
1.  null , if key is unique and added to the map  
2.  Old Value of the key , if key is duplicate

So , in HashSet add() method ,  we check the return value of map.put(key,value) method with null value



* How CuncurrentHashMap works?

**ConcurrentHashMap:** It allows concurrent access to the map. Part of the map called *Segment (internal data structure)*is only getting locked while adding or updating the map. So ConcurrentHashMap allows concurrent threads to read the value without locking at all. This data structure was introduced to improve performance.

1. ConcurrentHashMap allows multiple readers to read concurrently without any [blocking](http://javarevisited.blogspot.com/2012/02/what-is-blocking-methods-in-java-and.html).
2. This is achieved by partitioning Map into different parts based on concurrency level and locking only a portion of Map during updates.
3. Default concurrency level is 16, and accordingly Map is divided into 16 part and each part is governed with a different lock. This means, 16 thread can operate on Map simultaneously until they are operating on different part of Map. This makesConcurrentHashMap high performance despite keeping thread-safety intact.  Though, it comes with a caveat. Since update operations like put(), remove(), putAll() or clear() is not [synchronized](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html), concurrent retrieval may not reflect most recent change on Map.

1. In case of putAll() or clear(), which operates on whole Map, concurrent read may reflect insertion and removal of only some entries.
2. Iterator of ConcurrentHashMap's keySet area also [fail-safe](http://javarevisited.blogspot.in/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html) and doesn’t throwConcurrentModificationExceptoin..

Default concurrency level is 16 and can be changed, by providing a number which make sense and work for you while creating ConcurrentHashMap. Since concurrency level is used for internal sizing and indicate number of concurrent update without contention, so, if you just have few writers or thread to update Map keeping it low is much better. ConcurrentHashMap also uses ReentrantLock to internally lock its segments.

**When to use ConcurrentHashMap in Java?**

ConcurrentHashMap is best suited when you have multiple readers and few writers. If writers outnumber reader, or writer is equal to reader, than performance of ConcurrentHashMap effectively reduces.

**Summary**

Now we know What is ConcurrentHashMap in Java and when to use ConcurrentHashMap, it’s time to know and revise some important points about CHM in Java.

1. ConcurrentHashMap allows concurrent read and thread-safe update operation.

2. During the update operation, ConcurrentHashMap only locks a portion of Map instead of whole Map.

3. The concurrent update is achieved by internally dividing Map into the small portion which is defined by concurrency level.

4. Choose concurrency level carefully as a significantly higher number can be a waste of time and space and the lower number may introduce thread contention in case writers over number concurrency level.

5. All operations of ConcurrentHashMap are [thread-safe](http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html).

6. Since ConcurrentHashMap implementation doesn't lock whole Map, there is chance of read overlapping with update operations like put() and remove(). In that case result returned by get() method will reflect most recently completed operation from there start.

7. Iterator returned by ConcurrentHashMap is weekly consistent, [fail-safe](http://javarevisited.blogspot.com/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html) and never throw ConcurrentModificationException. In Java.

8. ConcurrentHashMap doesn't allow null as key or value.

9. During putAll() and clear() operations, the concurrent read may only reflect insertion or deletion of some entries.

* Internal Working of ArrayList in java?