Hashtable Vs SynchronizedMap Vs ConcurrentHashMap

September 21, 2017 Posted by Abhi Andhariya

Core Java, Core Java Interview Questions, Interview Questions, Java MultiThreading, Multithreading Interview Questions

Java Collection classes are heart of Java API. It is essential to use built-in java collections like HashMap, ArrayList or LinkedList for accessing, storing and processing data in java applications. For Example, we have extensively used HashMap to transfer data between two layers of our MVC framework.

In all core java interviews, you will definitely face questions on HashMap Internals like,

- what is the use of hashCode() and equals() method?
- How put() and get() method works in HashMap?

or you may face question like How is HashSet implemented internally in Java?

As a follow up question to above questions, interviewer may ask you : **is HashMap a thread-safe class?**

Well, **HashMap is not thread-safe.** If multiple threads are accessing the same HashMap object and try to modify the structure of the HashMap (using put() or remove() method), it may cause an inconsistency in the state of HashMap.

To use HashMap in multithreaded environment, you must write your relevant code inside synchronized block or use any external Lock implementation. But in that case there are high chances of errors and deadlock situations, if proper care has not been taken.

In short, it is not advisable to use HashMap in multithreaded environment. Instead use any of the similar thread-safe collections like Hashtable, Collections.SynchronizedMap or ConcurrentHashMap.

Though all of them are thread-safe, ConcurrentHashMap provides better performance than remaining two. lets understand them one by one.

Hashtable

Hashtable is a legacy class available since jdk 1.1 which uses synchronized methods to achieve thread safety. At a time only one thread can read or write into Hashtable. In other word, thread acquires lock on entire Hashtable instance. Hence its performance is quite slow and we can not utilize the advantages of multithreaded architecture.

Another point to note about Hashtable is that it does not allow null keys or values whereas HashMap allows one null key and any number of null values.

Collections.SynchronizedMap

SynchronizedMap is a static inner class of utility class java.util.Collections. It is quite similar to Hashtable and it also acquires lock on entire Map instance. It is not a legacy class like Hashtable and it was introduced in jdk 1.5.

It provides functionality to convert any thread-unsafe Map implementation to thread-safe implementation using Collections.synchronizedMap(Map map) static method. For Example,

```
import java.util.*;

public class SynchronizedMapDemo {
   public static void main(String[] args) {
   // create HashMap
   Map<String,String> map = new HashMap<String,String>();

   // populate the map
   map.put("1","Malay");
   map.put("2","Ankit");
   map.put("3","Chintan");

   // create a synchronized map
   Map<String,String> syncmap = Collections.synchronizedMap(map);

   System.out.println("Synchronized map is : "+syncmap);
   }
}
```

Output

```
Synchronized map is : {3=Chintan, 2=Ankit, 1=Malay}
```

It internally wraps all the methods of Map interface with synchronized keyword. For example, here is the internal put(), get() and remove() method implementation of SynchronizedMap :

```
private static class SynchronizedMap<K,V>
        implements Map<K,V>, Serializable {

                //other instance variables
                private final Map < k ,v > m;     // Backing Map
        final Object      mutex;        // Object on which to synchronize


        SynchronizedMap(Map<K,V> m) {
            if (m==null)
                throw new NullPointerException();
            this.m = m;
            mutex = this;
        }

                // Other constructors and methods

                public V get(Object key) {
            synchronized(mutex) {return m.get(key);}
        }

        public V put(K key, V value) {
            synchronized(mutex) {return m.put(key, value);}
        }
        public V remove(Object key) {
            synchronized(mutex) {return m.remove(key);}
        }
}
```

Click here to check full code of SynchronizedMap.

Similar to HashMap, Synchronized HashMap allows one null key and multiple null values as it just wraps the methods of HashMap with synchronized keyword. Rest of the behavior remains same as original collection.

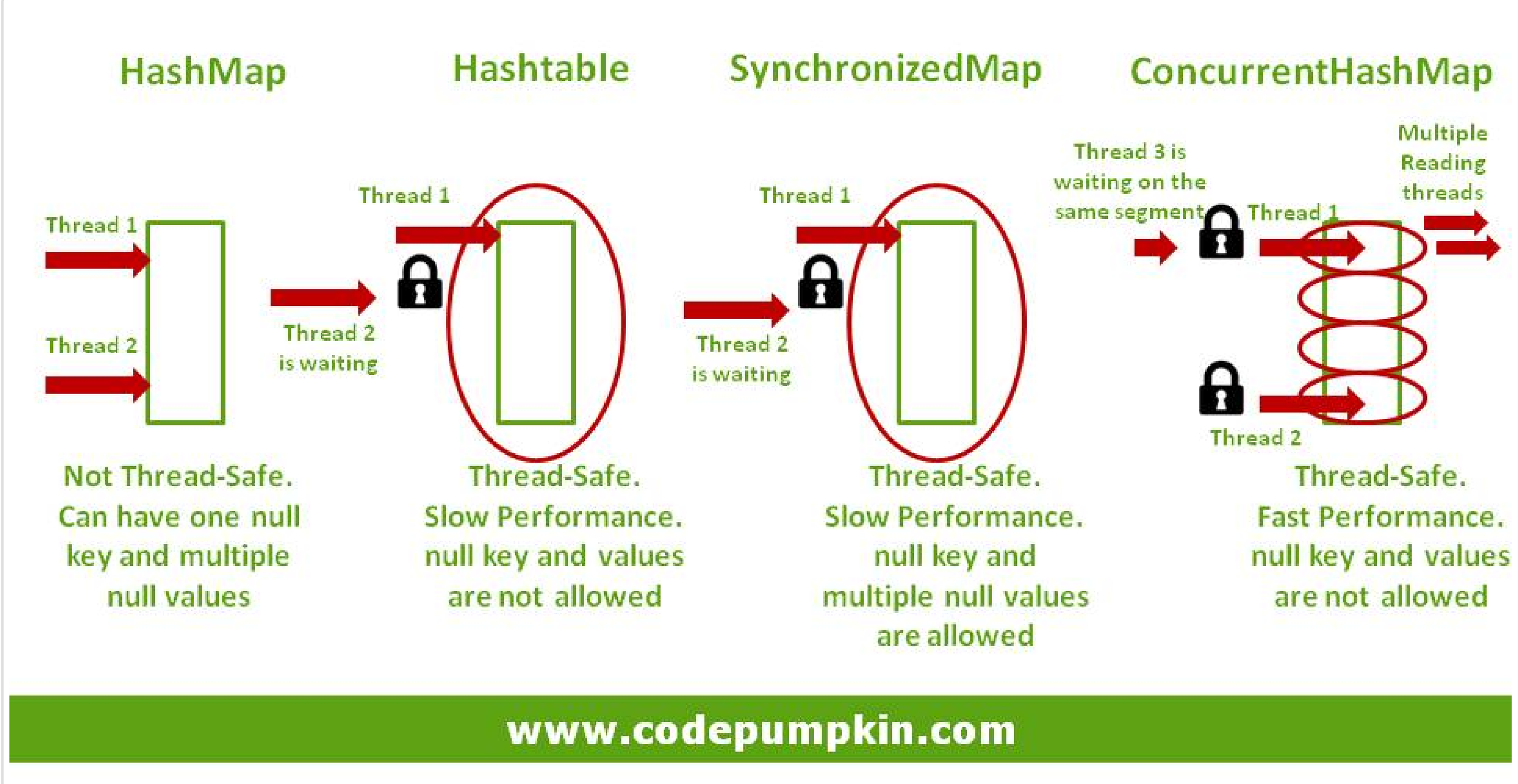
You can also check SynchronizedSortedMap which is similar data structure to convert thread-unsafe TreeMap and other SortedMap implementation to corresponding thread-safe collection.

ConcurrentHashMap

Hashtable and SynchronizedMap both acquires lock on entire Map object which provides thread-safety, but not good performance as at a time only one thread can access that Map instance.

To overcome this issue, ConcurrentHashMap was introduced in Java 5 along with other concurrent classes like CountDownLatch, CyclicBarrier, CopyOnWriteArrayList, BlockingQueue within java.util.Concurrent package.

More than one threads can read and write concurrently in ConcurrentHashMap and still it provides thread-safety. Amazing, isn' t it? How is it implemented internally?



Well, ConcurrentHashMap divides the Map instance into different segments. And each thread acquires lock on each segment. By default it allows 16 threads to access it simultaneously without any external synchronization i.e. by default concurrency level is 16. We can also increase or decrease the default concurrency level while creating ConcurrentHashMap by using below constructor :

```
ConcurrentHashMap(int initialCapacity, float loadFactor, int concurrencyLevel)
```

Can Multiple thread write in the same segment?

No. Thread acquires a lock on segment in put() operation and at a time only one thread can write in that segment.

Can two threads write in the different segment?

Yes. Two threads are allowed to write concurrently in different segments.

Can Multiple thread read from the same segment?

Yes. Thread doesn' t acquire a lock on segment in get() operation and any number of threads can read from the same segment.

If one thread is writing in a segment, can another thread read from that segment()?

Yes. but in this case last updated value will be seen by the reading thread.

Null keys and values

ConcurrentHashMap doesn' t allow null keys and null values.

Summary

1. HashMap is not thread-safe.
2. Once the size of Hashtable and SynchronizedMap becomes considerable large because for the iteration it has to be locked for the longer duration.
   While in ConcurrentHashMap, even if its size become very large, only portion or segment of the Map is locked which improves the performance in multithreading environment.
3. SynchronizedMap just wraps original Map object with synchronized block.
4. Hashtable and ConcurrentHashMap doesn' t allow null keys and null values, whereas SynchronizedMap may allow null keys and null values based on the original collection class being passed inside it.

We will soon publish our article on the internal working of ConcurrentHashMap. Stay tuned 🙂

You can also check our articles on sorting and searching such as Bubble Sort, Selection sort, Binary Search, Fibonacci Search, Merge Sort etc.

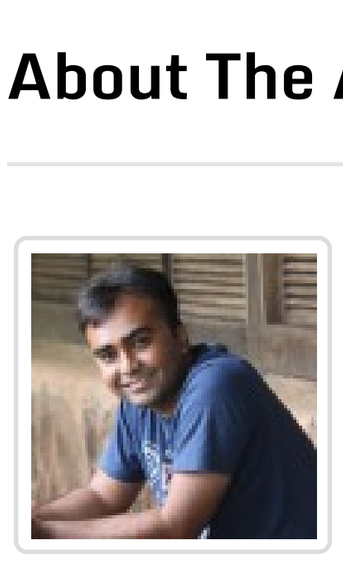
That' s all for this topic. If you guys have any suggestions or queries, feel free to drop a comment. We would be happy to add that in our post. You can also contribute your articles by creating contributor account here.

Happy Learning 🙂

If you like the content on CodePumpkin and if you wish to do something for the community and the planet Earth, you can donate to our campaign for planting more trees at CodePumpkin Cauvery Calling Campaign.

We may not get time to plant a tree, but we can definitely donate ₹42 per Tree.

About The Author



Abhi Andhariya Visit Full Profile

Surviving Java Developer, Passionate Blogger, Table Tennis Lover, Bookworm, Occasional illustrator and a big fan of Joey Tribbiani, The Walking Dead and Game of Thrones...!!

Tags: Collection Framework, Collections, Core Java, HashMap, Java, Synchronization, Thread, x vs y

Comments And Queries

If you want someone to read your code, please put the code inside <pre><code> and </code></pre> tags. For example:

```
<pre><code class="java">
String foo = "bar";
</code></pre>
```

For more information on supported HTML tags in disqus comment, click here.