**HashMap, HashTable and ConcurrentHashMap**

HashMap, HashTable and ConcurrentHashMap are three most important data structures of Java Collection Framework. All three hold the data in the form of key-value pairs. HashTable and ConcurrentHashMap are internally synchronized and hence thread safe. HashMap is not synchronized, but you can make it synchronized externally using Collections.synchronizedMap(). In this post, we will see the differences between externally synchronized HashMap Vs HashTable Vs ConcurrentHashMap in Java.

#### Differences Between Synchronized HashMap, HashTable And ConcurrentHashMap In Java :

**1) Locking Mechanism**

Synchronized HashMap and HashTable maintain object level lock. i.e the whole map is locked. Only one thread can enter into the map at any moment of time and others have to wait for the lock.

But in ConcurrentHashMap, the whole map is not locked. The map is divided into number of segments and each segment maintains its own lock. Any thread wants to enter into a segment have to acquire that segment’s lock.

This number of segments is decided by the parameter called Concurrency\_Level which is passed while instantiating ConcurrentHashMap. By default, it’s value is 16. That means, the map is divided into 16 segments and 16 threads can enter into the map at at time.

**2) Which operations are synchronized?**

In synchronized HashMap and HashTable, all operations are synchronized. That means, whatever the operation you want to perform on the map, whether it is read or update, you have to acquire object lock.

But in ConcurrentHashMap, only update operations are synchronized. Read operations are not synchronized. That means, any number of threads which wants to perform read operations on the map can enter into the map without waiting for the lock. Only update operations need segment level lock. Read operations don’t need any type of lock.

**3) How many threads can enter into a map at a time?**

Whatever may be the operation, only one thread can enter into a synchronized HashMap and Hashtable at any moment of time. Other threads have to wait for the lock.

On the other hand, minimum 16 threads can perform update operations on ConcurrentHashMap and any number of threads can perform read operations at a time without waiting for the lock.

**4) Null Keys And Null Values**

Synchronized HashMap allows one null key and any number of null values.

HashTable and ConcurrentHashMap doesn’t allow even a single null key and null values.

**5) Nature Of Iterators**

Iterators returned by synchronized HashMap are fail-fast in nature. i.e they throw ConcurrentModificationException if the map is modified after the creation of iterator.

Iterators returned by HashTable and ConcurrentHashMap are fail-safe in nature. i.e they don’t throw ConcurrentModificationException if the map is modified after the creation of iterator.

**6) When they are introduced?**

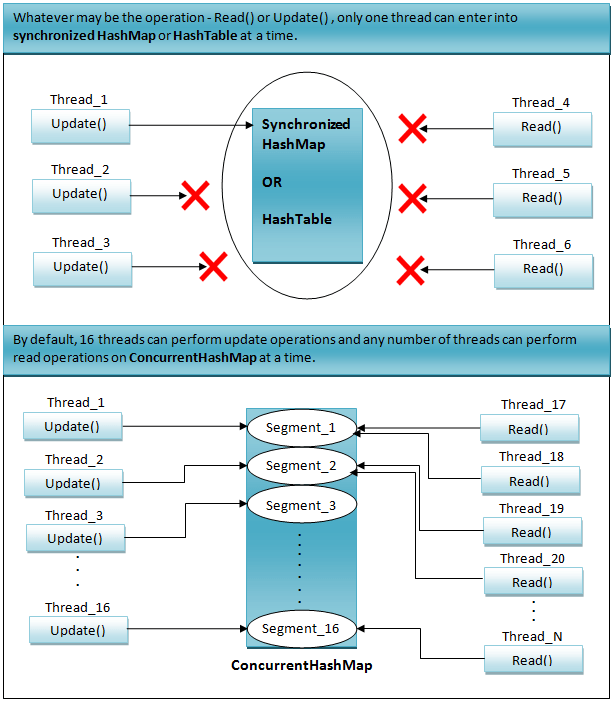
HashTable was there since JDK 1.1. From JDK 1.2, it has been made a part of Java Collection Framework.

HashMap is introduced in JDK 1.2 and ConcurrentHashMap in JDK 1.5.

**7) When to use what?**

HashTable is the legacy class. It is sometime considered as due for deprecation. So, it is recommended that not to use HashTable in your applications.

If you want high level of data consistency, then only consider using synchronized HashMap. Otherwise, in all the concurrently executing multi threaded circumstances, ConcurrentHashMap is the best choice.



**Synchronized HashMap Vs HashTable Vs ConcurrentHashMap In Java :**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Synchronized HashMap | HashTable | ConcurrentHashMap |
| Locking Level | Object Level | Object Level | Segment Level |
| Synchronized operations | All operations are synchronized. | All operations are synchronized. | Only update operations are synchronized. |
| How many threads can enter into a map at a time? | Only one thread | Only one thread | By default, 16 threads can perform update operations and any number of threads can perform read operations at a time. |
| Null Keys And Null Values | Allows one null key and any number of null values. | Doesn’t allow null keys and null values. | Doesn’t allow null keys and null values. |
| Nature Of Iterators | Fail-Fast | Fail-Safe | Fail-Safe |
| Introduced In? | JDK 1.2 | JDK 1.0 | JDK 1.5 |
| When To Use? | Use only when high level of data consistency is required in multi threaded environment. | Don’t Use. Not recommended as it is a legacy class. | Use in all multi threaded environment except where high level of data consistency is required. |