# Difference between Synchronized and Concurrent Collections in Java? Answer

## Synchronized vs Concurrent Collections

Though both Synchronized and Concurrent Collection classes provide thread-safety, the differences between them come in **performance**, **scalability,** and how they achieve thread-safety. Synchronized collections like

synchronized `HashMap`, `Hashtable`, `HashSet`, `Vector`, and synchronized `ArrayList` are much slower than their concurrent counterparts like `ConcurrentHashMap`, `CopyOnWriteArrayList`, and `CopyOnWriteHashSet`. The main reason for this slowness is **locking;** synchronized collections lock the whole collection e.g. whole Map or List while concurrent collection never locks the whole Map or List.

They achieve thread safety by using advanced and sophisticated techniques like lock stripping. For example, the ConcurrentHashMap divides the whole map into several segments and locks only the relevant segments, which allows multiple threads to access other segments of the same `ConcurrentHashMap` without locking.

Similarly, `CopyOnWriteArrayList` allows multiple reader threads to read without synchronization and when a write happens it copies the whole ArrayList and swaps with a newer one.

So if you use concurrent collection classes in their favorable conditions like for more reading and fewer updates, they are much more scalable than synchronized collections.

By the way, Collection classes are the heart of Java API though I feel using them judiciously is an art. It's my personal experience where I have improved performance by using ArrayList where legacy codes are unnecessarily used Vector etc. JDK 1.5 introduces some good concurrent collections which are highly efficient for high volume, low latency Java applications.

And, If you are new to the Java world then I also recommend you go through these **Java Collections Courses** from Pluralsight and Udemy to learn Java in a better and more structured way. This is one of the best and up-to-date courses to learn Java online.

## Synchronized Collections vs Concurrent Collections in Java

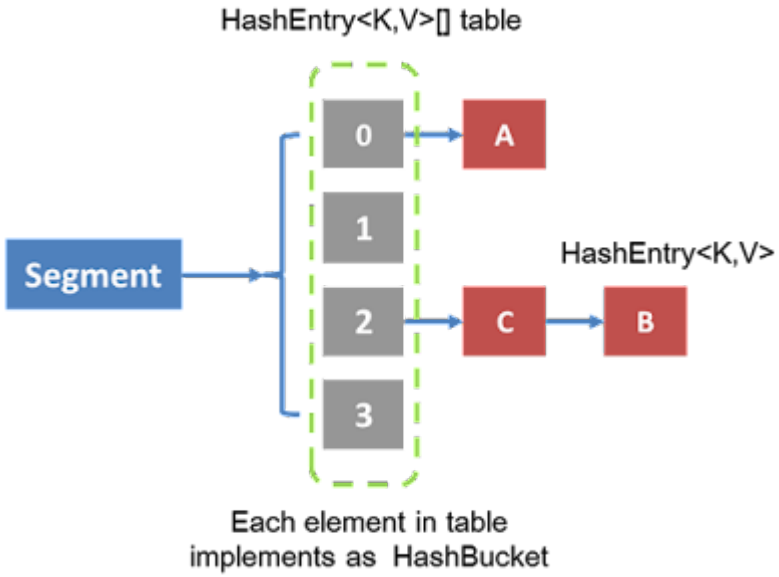
The synchronized collections classes, Hashtable and Vector, and the synchronized wrapper classes, `Collections.synchronizedMap()` and `Collections.synchronizedList()`, provides a basic conditionally thread-safe implementation of Map and List.

However, several factors make them unsuitable for use in highly concurrent applications, most importantly their single collection-wide lock is an impediment to scalability and it often becomes necessary to lock a collection for a considerable time during iteration to prevent ConcurrentModificationException.

The `ConcurrentHashMap` and `CopyOnWriteArrayList` implementations provide much higher concurrency and scalability while preserving the thread safety with some minor compromises in their promises to callers.

The `ConcurrentHashMap` and `CopyOnWriteArrayList` are not necessarily useful everywhere you might use HashMap or ArrayList but are designed to optimize specific common situations. Many concurrent applications will benefit from their use.

So what is the difference between Hashtable and ConcurrentHashMap, both can be used in a multi-threaded environment but once the size of Hashtable becomes considerably large performance degrades because for iteration it has to be locked for a longer duration?




Since ConcurrentHashMap introduced the concept of segmentation, It doesn't matter whether how large it becomes because only certain parts of it get locked to provide thread safety so many other readers can still access maps without waiting for iteration to complete.