Difference between Traditional Collections and Concurrent Collections in java

We all know about about Traditional Collections ( i.e. [List](https://www.geeksforgeeks.org/list-interface-java-examples/), [Set](https://www.geeksforgeeks.org/set-in-java/), [Queue](https://www.geeksforgeeks.org/queue-interface-java/) and its implemented Classes) and Concurrent Collection (i.e. ConcurrentMap interface, ConcurrentHashMap class, CopyOnWriteArrayList class etc). In these two Collections, there are few differences like:

* Most of the Classes which are present in **Traditional Collections (i.e [ArrayList](https://www.geeksforgeeks.org/arraylist-in-java/), [LinkedList](https://www.geeksforgeeks.org/linked-list-in-java/), [HashMap](https://www.geeksforgeeks.org/hashmap-treemap-java/) etc)** are non-synchronized in nature and Hence there is no thread-safety. But All the classes present in Concurrent Collections are synchronized in nature. Therefore In Concurrent classes, we dont have to take care about Thread-safety.
* While Traditional Collections also have **some classes (like**[**Vector**](https://www.geeksforgeeks.org/java-util-vector-class-java/)**,**[**Stack**](https://www.geeksforgeeks.org/stack-class-in-java/)**etc)** which are synchronized in nature and Traditional Collections also have **SynchronizedSet, SynchronizedList, SynchronizedMap** methods through which we can get Synchronized version of non-synchronized objects. But these above Synchronized classes are not good in terms of performance because of wide-locking mechanism .Whereas Concurrent Collections classes performance are relatively high than Traditional Collections classes.
* In the Traditional Collections, if a thread is iterating a Collection object and if another thread try to add new element in that iterating object simultaneously then we will get **RuntimeException ConcurrentModificationException**. Whereas In the above case, we will not get any Runtime Exception if we are Working with Concurrent Collections Classes.
* Traditional Collections classes is good choice if we are not dealing with thread in our application. whereas because of the Concurrent/Synchronized Collection we can use multiple Threads which are dealing with Collections Object. Therefore Concurrent Collections are best choice if we are dealing Multiple Threads in our application.

**1. ConcurrentHashMap**

ConcurrentHashMap is undoubtedly most popular collection class introduced in Java 5 and most of us are already using it. ConcurrentHashMap provides a concurrent alternative of [Hashtable or Synchronized Map](http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html) classes with aim to support higher level of concurrency by implementing fined grained locking. Multiple reader can access the Map concurrently  while a portion of Map gets locked for write operation depends upon concurrency level of Map. ConcurrentHashMap provides better scalability than there synchronized counter part. [Iterator](http://javarevisited.blogspot.com/2011/10/java-iterator-tutorial-example-list.html) of ConcurrentHashMap are [fail-safe iterators](http://javarevisited.blogspot.com/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html) which doesn't throw ConcurrencModificationException thus eliminates another requirement of locking during iteration which result in further scalability and performance.

**2. CopyOnWriteArrayList and CopyOnWriteArraySet**

CopyOnWriteArrayList is a concurrent alternative of synchronized List. CopyOnWriteArrayList provides better concurrency than [synchronized](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html)List by allowing multiple concurrent reader and replacing the whole list on write operation. Yes, write operation is costly on CopyOnWriteArrayList but it performs better when there are multiple reader and requirement of iteration is more than writing. Since CopyOnWriteArrayList Iterator also don't throw ConcurrencModificationException it eliminates need to lock the collection during iteration. Remember both ConcurrentHashMap and CopyOnWriteArrayList doesn't provides same level of locking as Synchronized Collection and achieves [thread-safety by](http://javarevisited.blogspot.com/2012/01/how-to-write-thread-safe-code-in-java.html) there locking and mutability strategy. So they perform better if requirements suits there nature. Similarly, CopyOnWriteArraySet is a concurrent replacement to Synchronized Set. See [What is CopyOnWriteArrayList in Java](http://java67.blogspot.com/2012/09/what-is-copyonwritearraylist-in-java-example-vs-arraylist.html) for more details

**3. BlockingQueue**

BlockingQueue is also one of better known collection class in Java 5. BlockingQueue makes it easy to implement [producer-consumer design pattern](http://javarevisited.blogspot.com/2012/02/producer-consumer-design-pattern-with.html) by providing inbuilt blocking support for put() and take() method. put() method will block if Queue is full while take() method will block if Queue is empty. Java 5 API provides two concrete implementation of BlockingQueue in form of [ArrayBlockingQueue and LinkedBlockingQueue](http://javarevisited.blogspot.com/2012/12/blocking-queue-in-java-example-ArrayBlockingQueue-LinkedBlockingQueue.html), both of them implement FIFO ordering of element. ArrayBlockingQueue is backed by Array and its bounded in nature while LinkedBlockingQueue is optionally bounded. Consider using BlockingQueue to solve producer Consumer problem in Java instead of writing your won [wait-notify code](http://java67.blogspot.com/2012/12/producer-consumer-problem-with-wait-and-notify-example.html). Java 5 also provides PriorityBlockingQueue, another implementation of BlockingQueue which is ordered on priority and useful if you want to process elements on order other than FIFO.

**4. Deque and BlockingDeque**

Deque interface is added in Java 6 and it extends Queue interface to support insertion and removal from both end of Queue referred as head and tail. Java6 also provides concurrent implementation of Deque like ArrayDeque and LinkedBlockingDeque. Deque Can be used efficiently to increase parallelism in program by allowing set of [worker thread](http://javarevisited.blogspot.sg/2013/01/threadlocal-memory-leak-in-java-web.html) to help each other by taking some of work load from other thread by utilizing Deque double end consumption property. So if all [Thread](http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html) has there own set of task Queue and they are consuming from head; helper thread can also share some work load via consumption from tail.

**5. ConcurrentSkipListMap and ConcurrentSkipListSet**

Just like [ConcurrentHashMap](http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html) provides a concurrent alternative of [synchronized HashMap](http://javarevisited.blogspot.com/2010/10/difference-between-hashmap-and.html). ConcurrentSkipListMap and ConcurrentSkipListSet provide concurrent alternative for synchronized version of SortedMap and SortedSet. For example instead of using TreeMap or TreeSet wrapped inside synchronized Collection, You can consider using ConcurrentSkipListMap or ConcurrentSkipListSet from java.util.concurrent package. They also implement NavigableMap and NavigableSet to add additional navigation method we have seen in our post [How to use NavigableMap in Java](http://javarevisited.blogspot.sg/2013/01/what-is-navigablemap-in-java-6-example-submap-head-tail.html).

That’s all on this list of concurrent Collection classes from Java 5 and 6. They are added on java.util.concurrent package as concurrent alternative of there synchronized counterpart. It’s good idea to learn these Collection classes along with other popular classes from Java Collection Framework.