**Concurrent Collection**

The concurrent API defines several collection classes that have been engineered for concurrent operation.

These offer concurrent alternatives to their related classes defined by the Collections Framework.

These collections work much like the other collections except that they provide concurrency support.

Concurrent collections (e.g., ConcurrentHashMap), achieve thread-safety by dividing their data into segments.

In a ConcurrentHashMap, for example, different threads **can acquire locks on each segment**, so multiple threads can access the Map at the same time (a.k.a. concurrent access).

Concurrent collections are much more performant than synchronized collections, due to the inherent advantages of concurrent thread access.

So, the choice of what type of thread-safe collection to use depends on the requirements of each use case, and it should be evaluated accordingly.

**What is difference between Traditional Collections and Concurrent Collections in java?**

**Answer: -** There are following difference between Traditional Collections and Concurrent Collections: -

* Most of the Classes which are present in Traditional Collections 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 don't have to take care about Thread-safety.
* While Traditional Collections also have some classes (like **Vector**, **Stack** 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 tries 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.

**What are the Interfaces coming under Concurrent Collections?**

**Answer: -** Following are the list of Interfaces which comes under Concurrent Collections: -

1. **BlockingQueue Interface.**
2. **BlockingDeque Interface.**
3. **TransferQueue Interface.**
4. **ConcurrentMap Interface.**
5. **ConcurrentNavigableMap Interface.**

**What are the Classes coming under Concurrent Collections?**

**Answer: -** Following are the list of Classes which comes under Concurrent Collections: -

1. **ArrayBlockingQueue Class**
2. **DelayQueue Class**
3. **LinkedBlockingQueue Class**
4. **PriorityBlockingQueue Class**
5. **SynchronousQueue Class**
6. **LinkedBlockingDeque Class**
7. **LinkedTransferQueue Class**
8. **ConcurrentHashMap Class**
9. **ConcurrentSkipListMap Class**
10. **CopyOnWriteArrayList Class**
11. **ConcurrentSkipListSet Class**