**Need of Concurrent Collections in java**

As we already know Collections which is nothing but collections of Objects where we deal with the Objects using some pre-defined methods. But There are several problems which occurs when we use Collections concept in multi-threading. The problems which occur while using Collections in Multi-threaded application:

* Most of the Collections classes objects (like ArrayList, [LinkedList](https://www.geeksforgeeks.org/linked-list-in-java/), [HashMap](http://www.geeksforgeeks.org/java-util-hashmap-in-java/) etc) are non-synchronized in nature i.e. multiple threads can perform on an object at a time simultaneously. Therefore, objects are not thread-safe.
* Very few Classes objects (like [Vector](https://www.geeksforgeeks.org/java-util-vector-class-java/), [Stack](https://www.geeksforgeeks.org/stack-class-in-java/), [HashTable](https://www.geeksforgeeks.org/java-util-hashtable-class-java/)) are synchronized in nature i.e. at a time only one thread can perform on an Object. But here the problem is performance is low because at a time single thread execute an object and rest thread has to wait.
* The main problem is when one thread is iterating a Collections object then if another thread can’t modify the content of the object. If another thread tries to modify the content of object, then we will get RuntimeException saying ***ConcurrentModificationException***.
* Because of the above reason Collections classes is not suitable or we can say that good choice for multi-threaded applications.

To overcome the above problem SUN Microsystem introduced a new feature in JDK 1.5Version, which is nothing but Concurrent

// Java program to illustrate Concurrent

// Collection need

import java.util.\*;

class ConcurrentDemo extends Thread {

static ArrayList l = new ArrayList();

public void run()

{

try {

Thread.sleep(2000);

}

catch (InterruptedException e) {

System.out.println("Child Thread" +

" going to add element");

}

// Child thread trying to add new

// element in the Collection object

l.add("D");

}

public static void main(String[] args)

throws InterruptedException

{

l.add("A");

l.add("B");

l.add("c");

// We create a child thread that is

// going to modify ArrayList l.

ConcurrentDemo t = new ConcurrentDemo();

t.start();

// Now we iterate through the ArrayList

// and get exception.

Iterator itr = l.iterator();

while (itr.hasNext()) {

String s = (String)itr.next();

System.out.println(s);

Thread.sleep(6000);

}

System.out.println(l);

}

}

|  |
| --- |
|  |

Output:  
  
***Exception in thread “main” java.util.ConcurrentModificationException***

**Difference between Traditional Collections and Concurrent Collections in java**

We all know 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 is 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.

|  |
| --- |
|  |
| // Java program to illustrate ConcurrentCollection uses  import java.util.concurrent.CopyOnWriteArrayList;  import java.util.\*;  class ConcurrentDemo extends Thread {  static CopyOnWriteArrayList l =  new CopyOnWriteArrayList();  public void run()  {  try {  Thread.sleep(2000);  }  catch (InterruptedException e) {  System.out.println("Child Thread"  + " going to add element");  }    // Child thread trying to add new  // element in the Collection object  l.add("D");  }    public static void main(String[] args)  throws InterruptedException  {  l.add("A");  l.add("B");  l.add("c");    // We create a child thread that is  // going to modify ArrayList l.  ConcurrentDemo t = new ConcurrentDemo();  t.start();    // Now we iterate through the ArrayList  // and get exception.  Iterator itr = l.iterator();  while (itr.hasNext()) {  String s = (String)itr.next();  System.out.println(s);  Thread.sleep(6000);  }  System.out.println(l);  }  } |

output:

* A
* B
* c