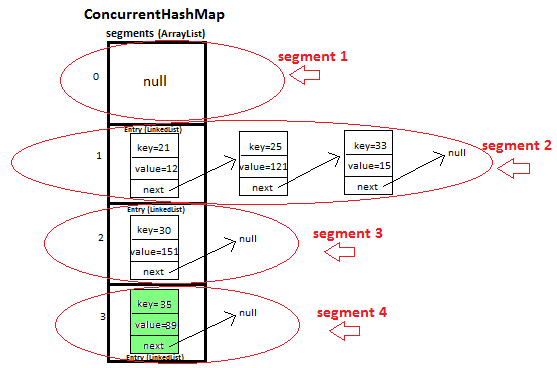
**Concurrent collections**

1. We know that most of the traditional Collections are not thread-safe

|  |  |
| --- | --- |
| Traditional Collections | Concurrent Collections |
| 1. Most of the traditional Collections are not thread-safe except Vector, Stack and Hashtable. | 1. All the classes are thread safe |
| 1. We have thread safe classes like Vector, Stack and Hashtable but for every operation the total object will be locked so other threads has to wait until the first thread completes it’s task. So performance is low | 1. We know that all these objects are thread safe and support bucket (segment) level locking. So while accessing elements of concurrent collection object the other threads can modify the object. So performance wise it is high. |
| 1. ConcurrentModifyicationException will occur if a thread try to modify the collection object while it is accessed by other thread. | 1. Won’t get ConcurrentModicationException |



**What are concurrent collections?**

|  |
| --- |
| Java 1.5 Concurrent package (`java.util.concurrent`) contains thread-safe collection classes that allow collections to be modified while iterating. By design Iterator implementation in `java.util` packages are fail-fast and throws ConcurrentModificationException. But Iterator implementation in `java.util.concurrent` packages are fail-safe and we can modify the collection while iterating. Some of these classes are `CopyOnWriteArrayList`, `ConcurrentHashMap`, `CopyOnWriteArraySet`.   1. Thread-safe 2. Bucket level locking will be supported 3. Never throw ConcurrentModificationException   Ex: CocurrentHashMap  Ex: CopyOnWriteArrayList  Ex: CopyOnWriteArraySet |

ExampleOn ConcurentModificationException

|  |
| --- |
| **package** app1;  **import** java.util.ArrayList;  **import** java.util.Vector;  **public** **class** ModifyExDemo  {  **static** Vector<Integer> *l1*=**new** Vector<Integer>();  //static ArrayList<Integer> l1=new ArrayList<Integer>();  **public** **static** **void** main(String[] args)**throws** Exception {  System.***out***.println(Thread.*currentThread*().getName()+"adding");  *l1*.add(100);  *l1*.add(200);  *l1*.add(300);  *l1*.add(400);  *l1*.add(500);  *l1*.add(600);    // **TODO** Auto-generated method stub  Thread t1=**new** Thread(()->{  **for**(Integer io:*l1*)  {  System.***out***.println(Thread.*currentThread*().getName()+"reading....");  System.***out***.println(io);  **try** {  System.***out***.println(Thread.*currentThread*().getName()+" sleepg");  Thread.*sleep*(1000);  }**catch**(Exception e)  {}  }  }  );  t1.start();  System.***out***.println("Started....");  System.***out***.println(Thread.*currentThread*().getName()+" Removing element ");  *l1*.remove(0);  }  }  Output:  mainadding  Started....  main Removing element  Exception in thread "Thread-0" java.util.ConcurrentModificationException  at java.base/java.util.Vector$Itr.checkForComodification(Vector.java:1292)  at java.base/java.util.Vector$Itr.next(Vector.java:1248)  at app1.ModifyExDemo.lambda$0(ModifyExDemo.java:19)  at java.base/java.lang.Thread.run(Thread.java:1583) |

Usage of CopyOnWriteArrayList

|  |
| --- |
| **package** app1;  **import** java.util.ArrayList;  **import** java.util.Vector;  **import** java.util.concurrent.CopyOnWriteArrayList;  **public** **class** ModifyExDemo  {  //static Vector<Integer> l1=new Vector<Integer>();  //static ArrayList<Integer> l1=new ArrayList<Integer>();  **static** CopyOnWriteArrayList<Integer> *l1*=**new** CopyOnWriteArrayList<Integer>();    **public** **static** **void** main(String[] args)**throws** Exception {  System.***out***.println(Thread.*currentThread*().getName()+"adding");  *l1*.add(100);  *l1*.add(200);  *l1*.add(300);  *l1*.add(400);  *l1*.add(500);  *l1*.add(600);    // **TODO** Auto-generated method stub  Thread t1=**new** Thread(()->{  **for**(Integer io:*l1*)  {  System.***out***.println(Thread.*currentThread*().getName()+"reading....");  System.***out***.println(io);  **try** {  System.***out***.println(Thread.*currentThread*().getName()+" sleepg");  Thread.*sleep*(1000);  }**catch**(Exception e)  {}  }  }  );  t1.start();  System.***out***.println("Started....");  System.***out***.println(Thread.*currentThread*().getName()+" Removing element ");  *l1*.remove(0);  }  }  Output:  Started....  main Removing element  Thread-0reading....  100  Thread-0 sleepg  Thread-0reading....  200  Thread-0 sleepg  Thread-0reading....  300  Thread-0 sleepg  Thread-0reading....  400  Thread-0 sleepg  Thread-0reading....  500  Thread-0 sleepg  Thread-0reading....  600  Thread-0 sleepg |

Example on CopyOnWriteSet

|  |
| --- |
| **package** app1;  **import** java.util.ArrayList;  **import** java.util.Vector;  **import** java.util.concurrent.CopyOnWriteArrayList;  **import** java.util.concurrent.CopyOnWriteArraySet;  **public** **class** ModifyExDemo  {  //static Vector<Integer> l1=new Vector<Integer>();  //static ArrayList<Integer> l1=new ArrayList<Integer>();  //static CopyOnWriteArrayList<Integer> l1=new CopyOnWriteArrayList<Integer>();  **static** CopyOnWriteArraySet<Integer> *l1*=**new** CopyOnWriteArraySet<Integer>();    **public** **static** **void** main(String[] args)**throws** Exception {  System.***out***.println(Thread.*currentThread*().getName()+"adding");  *l1*.add(100);  *l1*.add(200);  *l1*.add(300);  *l1*.add(400);  *l1*.add(500);  *l1*.add(600);    // **TODO** Auto-generated method stub  Thread t1=**new** Thread(()->{  **for**(Integer io:*l1*)  {  System.***out***.println(Thread.*currentThread*().getName()+"reading....");  System.***out***.println(io);  **try** {  System.***out***.println(Thread.*currentThread*().getName()+" sleepg");  Thread.*sleep*(1000);  }**catch**(Exception e)  {}  }  }  );  t1.start();  System.***out***.println("Started....");  System.***out***.println(Thread.*currentThread*().getName()+" Removing element ");  *l1*.remove(0);  }  }  Output:  mainadding  Started....  main Removing element  Thread-0reading....  100  Thread-0 sleepg  Thread-0reading....  200  Thread-0 sleepg  Thread-0reading....  300  Thread-0 sleepg  Thread-0reading....  400  Thread-0 sleepg  Thread-0reading....  500  Thread-0 sleepg  Thread-0reading....  600  Thread-0 sleepg |

What are fail fast iterator?

|  |
| --- |
| It is an iterator object which gets failed whenever you try to modify a traditional thread-safe Collection while reading elements by other thread. |

What are fail safe iterator?

|  |
| --- |
| It is an iterator object which doesn’t get failed whenever you try to modify a Concurrent Collection object while reading elements by other thread. |