CORE JAVA

With

SCJP / OCJP

Study Material

Chapter 14: Collections Framework

DURGA M.Tech

(Sun certified & Realtime Expert)

Ex. IBM Employee

Trained Lakhs of Students
for last 14 years across INDIA

India's No.1 Software Training Institute

DURGASOFT

www.durgasoft.com Ph: 9246212143 ,8096969696

Concurrent Collections (1.5)

Need fo Concurrent Collections
The Important Concurrent Classes
ConcurrentHashMap
CopyOnWriteArrayList
CopyOnWriteArraySet
ConcurrentMap (I)
ConcurrentHashMap
    Difference between HashMap and ConcurrentHashMap
    Difference between ConcurrentHashMap, synchronizedMap() and Hashtable
CopyOnWriteArrayList (C)
Differences between ArrayList and CopyOnWriteArrayList
Differences between CopyOnWriteArrayList, synchronizedList() and vector()
CopyOnWriteArraySet
Differences between CopyOnWriteArraySet() and synchronizedSet()
Fail Fast Vs Fail Safe Iterators
Differences between Fail Fast and Fail Safe Iterators
Enum with Collections
EnumSet
EnumMap
Queue
PriorityQueue
BlockingQueue
TransferQueue
Deque
BlockingDeque (I)

Need fo Concurrent Collections

- **Tradition Collection Object (Like ArrayList, HashMapEtc) can be accessed by Multiple Threads simultaneously and there May be a Chance of Data Inconsistency Problems and Hence these are Not Thread Safe.**
- **Already existing Thread Safe Collections (Vector, Hashtable, synchronizedList(), synchronizedSet(), synchronizedMap()) Performance wise Not Upto the Mark.**
- **Because for Every Operation Even for Read Operation Also Total Collection will be loaded by Only One Thread at a Time and it Increases waiting Time of Threads.**

```
importjava.util.ArrayList;
importjava.util.Iterator;
class Test {
        public static void main(String[] args) {
                ArrayList al = new ArrayList();
                al.add("A");
                al.add("B");
                al.add("C");
                Iterator itr = al.iterator();
                while (itr.hasNext()){
                        String s = (String)itr.next();
                        System.out.println(s);
                      //al.add("D");
                }
        }
}
```

```
A
B
C
```

```
A
RE: java.util.ConcurrentModificationException
```

- **Another Big Problem with Traditional Collections is while One Thread iterating Collection, the Other Threads are Not allowed to Modify Collection Object simultaneously if we are trying to Modify then we will get *ConcurrentModificationException.***
- **Hence these Traditional Collection Objects are Not Suitable for *Scalable Multi Threaded Applications.***
- **To Overcome these Problems SUN People introduced *Concurrent Collections* in 1.5 Version.**

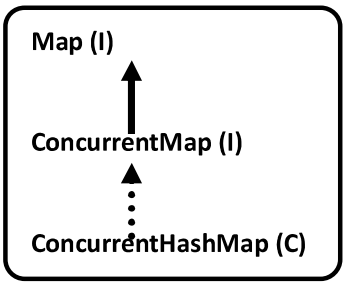
1) **Concurrent Collections are Always Thread Safe.**
2) **When compared with Traditional Thread Safe Collections Performance is More because of different Locking Mechanism.**
3) **While One Thread interacting Collection the Other Threads are allowed to Modify Collection in Safe Manner.**

Hence Concurrent Collections Never threw ConcurrentModificationException.

The Important Concurrent Classes are

- **ConcurrentHashMap**
- **CopyOnWriteArrayList**
- **CopyOnWriteArraySet**

ConcurrentMap (I):



Methods: It Defines the following 3 Specific Methods.

1) **Object putIfAbsent(Object Key, Object Value)**

To Add Entry to the Map if the specified Key is Not Already Available.

```
Object putIfAbsent(Object key, Object value)
if (!map.containsKey(key)) {
map.put(key, value);
}
else {
returnmap.get(key);
}
```

put()	putIfAbsent()
If the Key is Already Available, Old Value will be replaced with New Value and Returns Old Value.	If the Key is Already Present then Entry won't be added and Returns Old associated Value. If the Key is Not Available then Only Entry will be added.

```
importjava.util.concurrent.ConcurrentHashMap;
class Test {
        public static void main(String[] args) {
                ConcurrentHashMap m = new ConcurrentHashMap();
                m.put(101, "Durga");
                m.put(101, "Ravi");
                System.out.println(m); //{101=Ravi}
                m.putIfAbsent(101, "Siva");
                System.out.println(m); //{101=Ravi}
        }
}
```

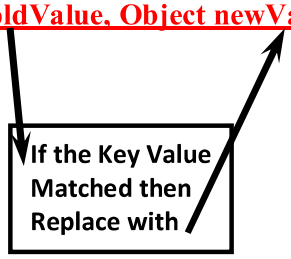
2) boolean remove(Object key, Object value)

Removes the Entry if the Key associated with specified Value Only.

```
if ( map.containsKey (key) &&map.get(key).equals(value) ) {
        map.remove(key);
        return true;
}
else {
return false;
}
```

```
importjava.util.concurrent.ConcurrentHashMap;
class Test {
        public static void main(String[] args) {
                ConcurrentHashMap m = new ConcurrentHashMap();
                m.put(101, "Durga");
                m.remove(101, "Ravi"); //Value Not Matched with Key So Nor Removed
                System.out.println(m); //{101=Durga}
                m.remove(101, "Durga");
                System.out.println(m); //{}
        }
}
```

3) **boolean replace(Object key, Object oldValue, Object newValue)**




```
if ( map.containsKey (key) &&map.get(key).equals(oldvalue) ) {
        map.put(key, newValue);
        return true;
}
else {
return false;
}
```

```
importjava.util.concurrent.ConcurrentHashMap;
class Test {
        public static void main(String[] args) {
                ConcurrentHashMap m = new ConcurrentHashMap();
                m.put(101, "Durga");
                m.replace(101, "Ravi", "Siva");
                System.out.println(m); //{101=Durga}
                m.replace(101, "Durga", "Ravi");
                System.out.println(m); //{101=Ravi}
        }
}
```

ConcurrentHashMap

- **Underlying Data Structure is Hashtable.**
- **ConcurrentHashMap allows Concurrent Read and Thread Safe Update Operations.**
- **To Perform Read Operation Thread won't require any Lock. But to Perform Update Operation Thread requires Lock but it is the Lock of Only a Particular Part of Map (Bucket Level Lock).**

- **Instead of Whole Map Concurrent Update achieved by Internally dividing Map into Smaller Portion which is defined by *Concurrency Level.***
- **The Default Concurrency Level is 16.**
- **That is ConcurrentHashMap Allows simultaneous Read Operation and simultaneously 16 Write (Update) Operations.**
- **null is Not Allowed for Both Keys and Values.**
- **While One Thread iterating the Other Thread can Perform Update Operation and ConcurrentHashMap Never throw *ConcurrentModificationException.***

Constructors:

1) **ConcurrentHashMap m = new ConcurrentHashMap();**
   Creates an Empty ConcurrentHashMap with Default Initial Capacity 16 and Default Fill Ratio
0.75and Default Concurrency Level 16.

2) **ConcurrentHashMap m = new ConcurrentHashMap(intinitialCapacity);**
3) **ConcurrentHashMap m = new ConcurrentHashMap(intinitialCapacity, float fillRatio);**
4) **ConcurrentHashMap m = new ConcurrentHashMap(intinitialCapacity, float fillRatio, intconcurrencyLevel);**
5) **ConcurrentHashMap m = new ConcurrentHashMap(Map m);**


```
importjava.util.concurrent.ConcurrentHashMap;
class Test {
        public static void main(String[] args) {
                ConcurrentHashMap m = new ConcurrentHashMap();
                m.put(101, "A");
                m.put(102, "B");
                m.putIfAbsent(103, "C");
                m.putIfAbsent(101, "D");
                m.remove(101, "D");
                m.replace(102, "B", "E");
                System.out.println(m); //{103=C, 102=E, 101=A}
        }
}
```

```
importjava.util.concurrent.ConcurrentHashMap;
importjava.util.*;
classMyThread extends Thread {
//static HashMap m = new HashMap(); // java.util.ConcurrentModificationException
staticConcurrentHashMap m = new ConcurrentHashMap();
        public void run() {
                try {
                        Thread.sleep(2000);
                }
                catch (InterruptedException e) {}
System.out.println("Child Thread updating Map");
                m.put(103, "C");
        }
        public static void main(String[] args) throws InterruptedException {
                m.put(101, "A");
                m.put(102, "B");
                MyThread t = new MyThread();
                t.start();
                Set s = m.keySet();
                Iterator itr = s.iterator();
                while (itr.hasNext())     {
                        Integer I1 = (Integer) itr.next();
                SOP("Main Thread iterating and Current Entry is:"+I1+"....."+m.get(I1));
                        Thread.sleep(3000);
                }
                System.out.println(m);
        }
}
```

```
Main Thread iterating and Current Entry is:102.....B
Child Thread updating Map
Main Thread iterating and Current Entry is:101.....A
{103=C, 102=B, 101=A}
```

Update and we won't get any *ConcurrentModificationException.*

- If we Replace ConcurrentHashMap with HashMap then we will get *ConcurrentModificationException.*

```
importjava.util.Iterator;
class Test {
        public static void main(String[] args) throws InterruptedException {
                ConcurrentHashMap m = new ConcurrentHashMap();
                m.put(101, "A");
                m.put(102, "B");
                Iterator itr = m.keySet().iterator();
                m.put(103, "C");
                while (itr.hasNext())     {
                        Integer I1 = (Integer) itr.next();
                        System.out.println(I1+"....."+m.get(I1));
                        Thread.sleep(3000);
                }
                System.out.println(m);
                }
}
```

```
102.....B
101.....A
{103=C, 102=B, 101=A}
```


DURGASOFT, # 202,2nd Floor,HUDA Maithrivanam,Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 9246212143 | www.durgasoft.com

Reason:

- **In the Case of ConcurrentHashMap iterator creates a Read Only Copy of Map Object and iterates over that Copy if any Changes to the Map after getting iterator it won't be affected/ reflected.**
- **In the Above Program if we Replace ConcurrentHashMap with HashMap then we will get ConcurrentModificationException.**

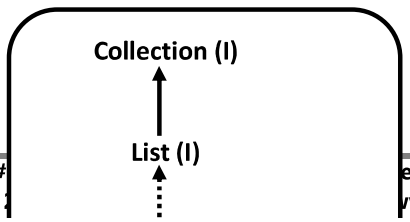
Difference between HashMap and ConcurrentHashMap

HashMap	ConcurrentHashMap
It is Not Thread Safe.	It is Thread Safe.
Relatively Performance is High because Threads are Not required to wait to Operate on HashMap.	Relatively Performance is Low because Some Times Threads are required to wait to Operate on ConcurrentHashMap.
While One Thread iterating HashMap the Other Threads are Not allowed to Modify Map Objects Otherwise we will get Runtime Exception Saying ConcurrentModificationException.	While One Thread iterating ConcurrentHashMap the Other Threads are allowed to Modify Map Objects in Safe Manner and it won't throw ConcurrentModificationException.
Iterator of HashMap is Fail-Fast and it throws ConcurrentModificationException.	Iterator of ConcurrentHashMap is Fail-Safe and it won't throwsConcurrentModificationException.
null is allowed for Both Keys and Values.	null is Not allowed for Both Keys and Values. Otherwise we will get NullPointerException.
Introduced in 1.2 Version.	Introduced in 1.5 Version.

Difference between ConcurrentHashMap, synchronizedMap() and Hashtable

ConcurrentHashMap	synchronizedMap()	Hashtable
We will get Thread Safety without locking Total Map Object Just with Bucket Level Lock.	We will get Thread Safety by locking Whole Map Object.	We will get Thread Safety by locking Whole Map Object.
At a Time Multiple Threads are allowed to Operate on Map Object in Safe Manner.	At a Time Only One Thread is allowed to Perform any Operation on Map Object.	At a Time Only One Thread is allowed to Operate on Map Object.
Read Operation can be performed without Lock but write Operation can be performed with Bucket Level Lock.	Every Read and Write Operations require Total Map Object Lock.	Every Read and Write Operations require Total Map Object Lock.
While One Thread iterating Map Object, the Other Threads are allowed to Modify Map and we won't get ConcurrentModificationException.	While One Thread iterating Map Object, the Other Threads are Not allowed to Modify Map. Otherwise we will get ConcurrentModificationException	While One Thread iterating Map Object, the Other Threads are Not allowed to Modify Map. Otherwise we will get ConcurrentModificationException
Iterator of ConcurrentHashMap is Fail-Safe and won't raise ConcurrentModificationException.	Iterator of synchronizedMap is Fail-Fast and it will raise ConcurrentModificationException.	Iterator of synchronizedMap is Fail-Fast and it will raise ConcurrentModificationException.
null is Not allowed for Both Keys and Values.	null is allowed for Both Keys and Values.	null is Not allowed for Both Keys and Values.
Introduced in 1.5 Version.	Introduced in 1.2 Version.	Introduced in 1.0 Version.

CopyOnWriteArrayList (C):



- **It is a Thread Safe Version of ArrayList as the Name indicates CopyOnWriteArrayList Creates a Cloned Copy of Underlying ArrayList for Every Update Operation at Certain Point Both will Synchronized Automatically Which is taken Care by JVM Internally.**
- **As Update Operation will be performed on cloned Copy there is No Effect for the Threads which performs Read Operation.**
- **It is Costly to Use because for every Update Operation a cloned Copy will be Created. Hence CopyOnWriteArrayList is the Best Choice if Several Read Operations and Less Number of Write Operations are required to Perform.**
- **Insertion Order is Preserved.**
- **Duplicate Objects are allowed.**
- **Heterogeneous Objects are allowed.**
- **null Insertion is Possible.**
- **It implements Serializable, Clonable and RandomAccess Interfaces.**
- **While One Thread iterating CopyOnWriteArrayList, the Other Threads are allowed to Modify and we won't get ConcurrentModificationException. That is iterator is Fail Safe.**
- **Iterator of ArrayList can Perform Remove Operation but Iterator of CopyOnWriteArrayList can't Perform Remove Operation. Otherwise we will get RuntimeException Saying UnsupportedOperationException.**

Constructors:

1) **CopyOnWriteArrayList l = new CopyOnWriteArrayList();**
2) **CopyOnWriteArrayList l = new CopyOnWriteArrayList(Collection c);**
3) **CopyOnWriteArrayList l = new CopyOnWriteArrayList(Object[] a);**

Methods:

1. **booleanaddIfAbsent(Object o): The Element will be Added if and Only if List doesn't contain this Element.**

```
CopyOnWriteArrayList l = new CopyOnWriteArrayList();
l.add("A");
l.add("A");
l.addIfAbsent("B");
l.addIfAbsent("B");
System.out.println(l); //[A, A, B]
```

2. **ntaddAllAbsent(Collection c): The Elements of Collection will be Added to the List if Elements are Absent and Returns Number of Elements Added.**

```
ArrayList l = new ArrayList();
l.add("A");
l.add("B");

CopyOnWriteArrayList l1 = new CopyOnWriteArrayList();
l1.add("A");
l1.add("C");
System.out.println(l1); //[A, C]
l1.addAll(l);
System.out.println(l1); //[A, C, A, B]

ArrayList l2 = new ArrayList();
l2.add("A");
l2.add("D");
l1.addAllAbsent(l2);
System.out.println(l1); //[A, C, A, B, D]
```

```
importjava.util.concurrent.CopyOnWriteArrayList;
importjava.util.ArrayList;
class Test {
        public static void main(String[] args) {
ArrayList l = new ArrayList();
                l.add("A");
                l.add("B");

                CopyOnWriteArrayList l1 = new CopyOnWriteArrayList();
l1.addIfAbsent("A");
l1.addIfAbsent("C");
                l1.addAll(l);

ArrayList l2 = new ArrayList();
                l2.add("A");
        l2.add("E");
                l1.addAllAbsent(l2);

                System.out.println(l1); //[A, C, A, B, E]
}
}
```

```
importjava.util.concurrent.CopyOnWriteArrayList;
importjava.util.*;
classMyThread extends Thread {
staticCopyOnWriteArrayList l = new CopyOnWriteArrayList();
        public void run() {
                try {   Thread.sleep(2000);   }
                catch (InterruptedException e) {}
System.out.println("Child Thread Updating List");
                l.add("C");
        }
```

- **In the Above Example while Main Thread iterating List Child Thread is allowed to Modify and we won't get any ConcurrentModificationException.**
- **If we Replace CopyOnWriteArrayList with ArrayList then we will get ConcurrentModificationException.**
- **Iterator of CopyOnWriteArrayList can't Perform Remove Operation. Otherwise we will get RuntimeException: UnsupportedOperationException.**

```
importjava.util.concurrent.CopyOnWriteArrayList;
importjava.util.Iterator;
class Test {
public static void main(String[] args){
        CopyOnWriteArrayList l = new CopyOnWriteArrayList();
        l.add("A");
        l.add("B");
        l.add("C");
        l.add("D");
        System.out.println(l); //[A, B, C, D]
        Iterator itr = l.iterator();
        while (itr.hasNext()) {
                String s = (String)itr.next();
                if (s.equals("D"))
                        itr.remove();
        }
        System.out.println(l); //RE: java.lang.UnsupportedOperationException
}
}
```

- **If we Replace CopyOnWriteArrayList with ArrayList we won't get any UnsupportedOperationException.**
- **In this Case the Output is**
  - **[A, B, C, D]**
  - **[A, B, C]**

```
importjava.util.concurrent.CopyOnWriteArrayList;
importjava.util.Iterator;
class Test {
public static void main(String[] args) {
        CopyOnWriteArrayList l = new CopyOnWriteArrayList();
        l.add("A");
        l.add("B");
        l.add("C");
        Iterator itr = l.iterator();
        l.add("D");
        while (itr.hasNext()) {
                String s = (String)itr.next();
                System.out.println(s);
        }
}
}
```

```
A
B
C
```

Reason:

- **Every Update Operation will be performed on Separate Copy Hence After getting iterator if we are trying to Perform any Modification to the List it won't be reflected to the iterator.**

- **In the Above Program if we ReplaceCopyOnWriteArrayList with ArrayList then we will get RuntimeException: java.util.ConcurrentModificationException.**

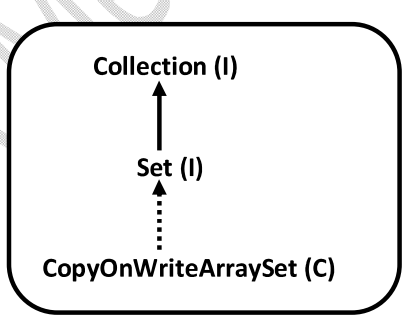
Differences between ArrayList and CopyOnWriteArrayList

ArrayList	CopyOnWriteArrayList
It is Not Thread Safe.	It is Not Thread Safe because Every Update Operation will be performed on Separate cloned Coy.
While One Thread iterating List Object, the Other Threads are Not allowed to Modify List Otherwise we will get ConcurrentModificationException.	While One Thread iterating List Object, the Other Threads are allowed to Modify List in Safe Manner and we won't get ConcurrentModificationException.
Iterator is Fail-Fsat.	Iterator is Fail-Safe.
Iterator of ArrayList can Perform Remove Operation.	Iterator of CopyOnWriteArrayList can't Perform Remove Operation Otherwise we will get RuntimeException: UnsupportedOperationException.
Introduced in 1.2 Version.	Introduced in 1.5 Version.

Differences between CopyOnWriteArrayList, synchronizedList() and vector()

CopyOnWriteArrayList	synchronizedList()	vector()
We will get Thread Safety because Every Update Operation will be performed on Separate cloned Copy.	We will get Thread Safety because at a Time List can be accessed by Only One Thread at a Time.	We will get Thread Safety because at a Time Only One Thread is allowed to Access Vector Object.
At a Time Multiple Threads are allowed to Access/ Operate on CopyOnWriteArrayList.	At a Time Only One Thread is allowed to Perform any Operation on List Object.	At a Time Only One Thread is allowed to Operate on Vector Object.
While One Thread iterating List Object, the Other Threads are allowed to Modify Map and we won't get ConcurrentModificationException.	While One Thread iterating , the Other Threads are Not allowed to Modify List. Otherwise we will get ConcurrentModificationException	While One Thread iterating, the Other Threads are Not allowed to Modify Vector. Otherwise we will get ConcurrentModificationException
Iterator is Fail-Safe and won't raise ConcurrentModificationException.	Iterator is Fail-Fast and it will raise ConcurrentModificationException.	Iterator is Fail-Fast and it will raise ConcurrentModificationException.
Iterator can't Perform Remove Operation Otherwise we will get UnsupportedOperationException.	Iterator canPerform Remove Operation.	Iterator can Perform Remove Operation.
Introduced in 1.5 Version.	Introduced in 1.2 Version.	Introduced in 1.0 Version.

CopyOnWriteArraySet :



- It is a Thread Safe Version of Set.
- Internally Implement by CopyOnWriteArrayList.
- Insertion Order is Preserved.
- Duplicate Objects are Notallowed.
- Multiple Threads can Able to Perform Read Operation simultaneously but for Every Update Operation a Separate cloned Copy will be Created.
- As for Every Update Operation a Separate cloned Copy will be Created which is Costly Hence if Multiple Update Operation are required then it is Not recommended to Use CopyOnWriteArraySet.

- **While One Thread iterating Set the Other Threads are allowed to Modify Set and we won't get ConcurrentModificationException.**
- **Iterator of CopyOnWriteArraySet can PerformOnly Read Operation and won't Perform Remove Operation. Otherwise we will get RuntimeException: UnsupportedOperatonException.**

Constructors:

1) CopyOnWriteArraySets = new CopyOnWriteArraySet();
Creates an Empty CopyOnWriteArraySet Object.

2) CopyOnWriteArraySet s = new CopyOnWriteArraySet(Collection c);
Creates CopyOnWriteArraySet Object which is Equivalent to given Collection Object.

Methods:Whatever Methods Present in Collection and Set Interfaces are the Only Methods Applicable for CopyOnWriteArraySet and there are No Special Methods.

```
importjava.util.concurrent.CopyOnWriteArraySet;
class Test {
public static void main(String[] args) {
CopyOnWriteArraySet s = new CopyOnWriteArraySet();
                s.add("A");
                s.add("B");
                s.add("C");
                s.add("A");
                s.add(null);
                s.add(10);
                s.add("D");
```

Differences between CopyOnWriteArraySet() and synchronizedSet()

CopyOnWriteArraySet()	synchronizedSet()
It is Thread Safe because Every Update Operation will be performed on Separate Cloned Copy.	It is Thread Safe because at a Time Only One Thread can Perform Operation.
While One Thread iterating Set, the Other Threads are allowed to Modify and we won't get ConcurrentModificationException.	While One Thread iterating, the Other Threads are Not allowed to Modify Seta Otherwise we will get ConcurrentModificationException.
Iterator is Fail Safe.	Iterator is Fail Fast.

Iterator can Perform Only Read Operation and can't Perform Remove Operation Otherwise we will get RuntimeException Saying UnsupportedOperationException.	Iterator can Perform Both Read and Remove Operations.
Introduced in 1.5 Version.	Introduced in 1.7 Version.

Fail Fast Iterator

Fail Fast Vs Fail Safe Iterators:

Fail Fast Iterator: While One Thread iterating Collection if Other Thread trying to Perform any Structural Modification to the underlying Collection then immediately Iterator Fails by raising ConcurrentModificationExcepion. Such Type of Iterators are Called Fail Fast Iterators.

```
importjava.util.ArrayList;
importjava.util.Iterator;
class Test {
public static void main(String[] args) {
ArrayList l = new ArrayList();
l.add("A");
l.add("B");
Iterator itr = l.iterator();
while(itr.hasNext()) {
                String s = (String)itr.next();
                System.out.println(s); //A
        l.add("C"); // java.util.ConcurrentModificationException
}
    }
}
```

Fail Fast Iterator

Note: Internally Fail Fast Iterator will Use Some Flag named with MOD to Check underlying Collection is Modified OR Not while iterating.

Fail Safe Iterator:

- While One Thread iterating if the Other Threads are allowed to Perform any Structural Changes to the underlying Collection, Such Type of Iterators are Called Fail Safe Iterators.
- Fail Safe Iterators won't raise ConcurrentModificationException because Every Update Operation will be performed on Separate cloned Copy.

```
importjava.util.concurrent.CopyOnWriteArraySet;
importjava.util.Iterator;
class Test {
public static void main(String[] args) {
CopyOnWriteArraySet l = new CopyOnWriteArraySet();
l.add("A");
l.add("B");
Iterator itr = l.iterator();
while(itr.hasNext()) {
        String s = (String)itr.next();
        System.out.println(s); //A
        l.add("C");
}
    }
}
```

Fail Safe Iterator

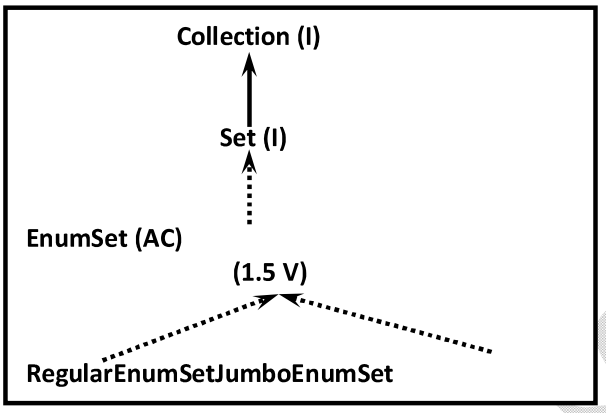
A
B

d - 500038,
om

Differences between Fail Fast and Fail Safe Iterators:

Property	Fail Fast	Fail Safe
Does it through ConcurrentModificationException?	Yes	No
Is the Cloned Copy will be Created?	No	Yes
Memory Problems	No	Yes
Examples	ArrayList, Vector, HashMap, HashSet	ConcurrentHashMap, CopyOnWriteArrayList, CopyOnWriteArraySet

Enum with Collections



EnumSet:

- Ii is a specially designed Set implemented Collection Applicable Only for Enum.
- Introduced in 1.5 Version.
- EnumSet is Internally implemented as Bit Vectors which Improves Performance Internally.
- The Performance of EnumSet is Very High if we want to Store Enum Constants than Traditional Collections (Like HashSet, LinkedHashSetEtc).
- All Elements of the EnumSet should be from Same Enum Type Only if we are trying to Add Elements from different enums then we will get Compile Time Error (i.e. EnumSet is Type Safe Collection).
- Iterator Returned by EnumSet Traverse, Iterate Elements in their Natural Order i.e. the Order in which the Enum Constants are declared i.e. the Order Returned by ordinal().
- Enum Iterator Never throwConcurrentModificationException.
- Inside EnumSet we can't Add null Otherwise we will get NullPointerException.
- EnumSet is an Abstract Class and Hence we can't Create Object directly by using new Key Word.
- EnumSet defined Several Factory Methods to Create EnumSet Object.
- EnumSet defines 2 Child Classes.
  - RegularEnumSet
  - JumboEnumSet
- The Factory Methods will Return this Class Objects Internally Based on Size if the Size is < 64 then RegularEnumSet will be choosed Otherwise if Size > 64 then JumboEnumSet will be choosed.

EnumMap:

- Ii is a specially designed Map to Use Enum Type Objects as Keys.
- Introduced in 1.5 Version.
- It implements *Serializable* and *Cloneable* Interfaces.
- EnumMap is Internally implemented by using Bit Vectors (Arrays), which Improves Performance when compared with Traditional Map Object Like HashMap Etc.
- All Keys to the EnumMap should be from a Single Enum if we are trying to Use from different Enum then we will get Compile Time Error. Hence EnumMap is Type Safe.
- Iterator Never throwConcurrentModificationException.

JAVA Means DURGASOFT

- **Iterators of EnumMap iterate Elements according to Ordinal Value of Enum Keys i.e. in which Order Enum Constants are declared in the Same Order Only Iterator will be iterated.**
- **null Key is Not allowed Otherwise we will get NullPointerException.**

Constructors

1) EnumMap m = new EnumMap(Class KeyType)
Creates an Empty EnumMap with specified Key Type.

2) EnumMap m = new EnumMap(EnumMap m1)
Creates an EnumMap with the Same Key Type and the specified EnumMap.Internally containing Same Mappings.

3) EnumMap m = new EnumMap(Map m1)
To Create and Equivalent EnumMap for given Map.

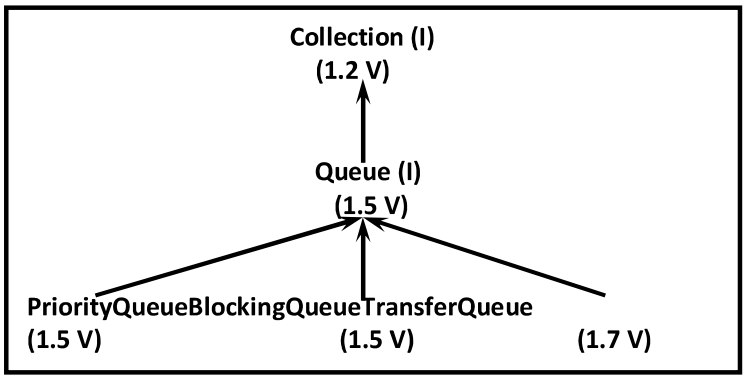
Methods:

EnumMap doesn't contain any New Methods. We have to Use General Map Methods Only.

```
importjava.util.*;
enum Priority {
        LOW, MEDIUM, HIGH
}
classEnumMapDemo {
public static void main(String[] args) {
EnumMap<Priority, String> m = new EnumMap<Priority, String> (Priority.class);
m.put(Priority.LOW, "24 Hours Response Time");
        m.put(Priority.MEDIUM, "3 Hours Response Time");
        m.put(Priority.HIGH, "1 Hour Response Time");
        System.out.println(m);
        Set s = m.keySet();
        Iterator<Priority>itr = s.iterator();
        while(itr.hasNext()) {
            Priority p = itr.next();
        System.out.println(p+"....."+m.get(p));
        }
    }
}
```

```
{LOW=24 Hours Response Time, MEDIUM=3 Hours Response Time, HIGH=1 Hour Response Time}
LOW.....24 Hours Response Time
MEDIUM.....3 Hours Response Time
HIGH.....1 Hour Response Time
```

Overview of java Queues



Queue:

If we want to Represent a Group of Individual Objects Prior to processing then Use should go for Queue.

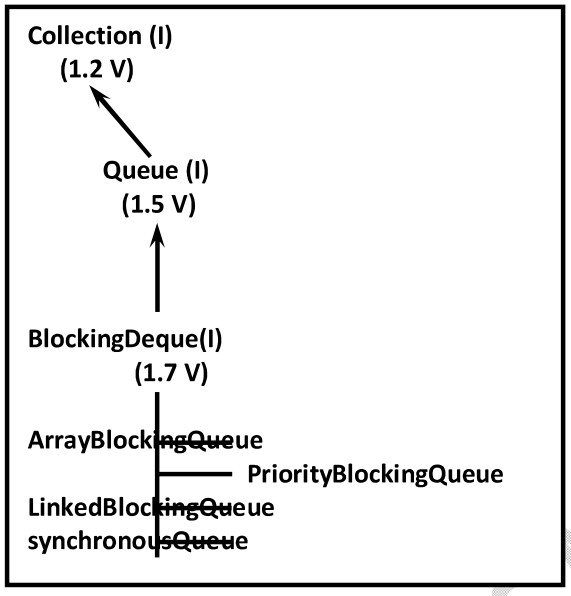
- **Queue is Child Interface of Collection.**

PriorityQueue:

- **It is the Implementation Class of Queue.**
- **If we want to Represent a Group of Individual Objects Prior to processing according to Priority then we should go for PriorityQueue.**

BlockingQueue:

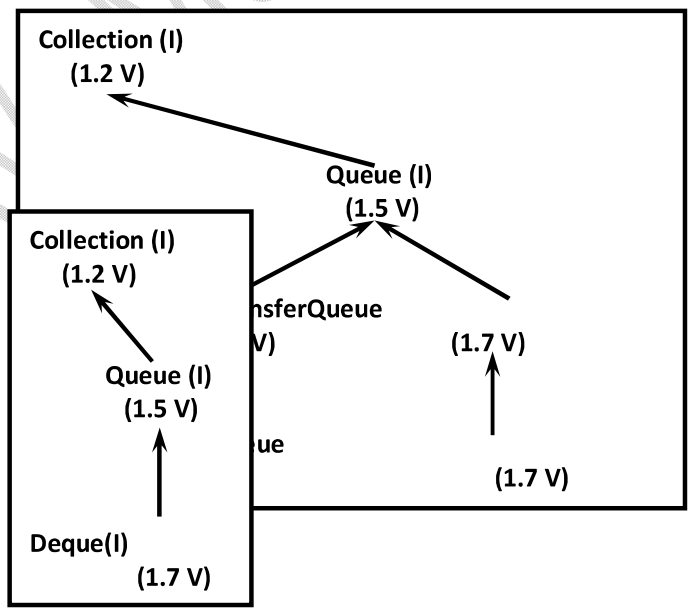
- **It is the Child Interface of Queue. Present in java.util.Concurrent Package.**
- **It is a Thread Safe Collection.**
- **It is a specially designed Collection Not Only to Store Elements but also Supports Flow Control by Blocking Mechanism.**
- **If Queue is Empty take() (Retrieval Operation) will be Blocked until Queue will be Updated with Items.**
- **put() will be blocked if Queue is Full until Space Availability.**
- **This Property Makes BlockingQueue Best Choice for Producer Consumer Problem. When One Thread producing Items to the Queue and the Other Thread consuming Items from the Queue.**



TransferQueue:

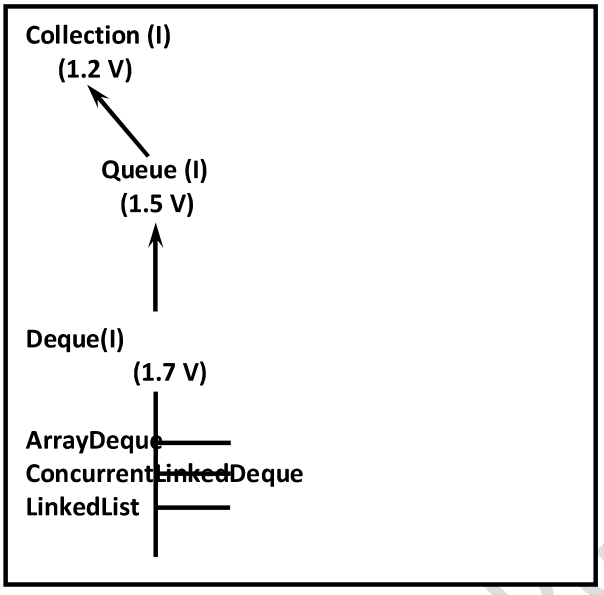
- In BlockingQueue we can Only Put Elements into the Queue and if Queue is Full then Our put() will be blocked until Space is Available.
- But in TransferQueue we can also Block until Other Thread receiving Our Element. Hence this is the Behavior of transfer().
- In BlockingQueue we are Not required to wait until Other Threads Receive Our Element but in TransferQueue we have to wait until Some Other Thread Receive Our Element.
- TrasferQueue is the Best Choice for Message Passing Application where Guarantee for the Delivery.

Deque (I)



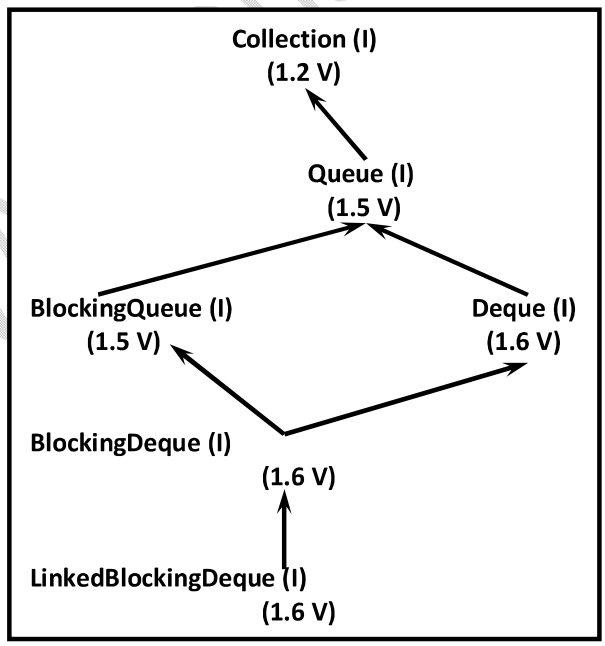
DURGASOFT, # 202,2nd Floor,HUDA Maitrivanam,Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 9246212143 | www.durgasoft.com

- **It Represents a Queue where we can Insert and Remove Elements from Deque, Both Ends of Queue i.e. Deque Means Double Ended Queue.**
- **It is Also pronounced as Deck Like Deck of Cards.**

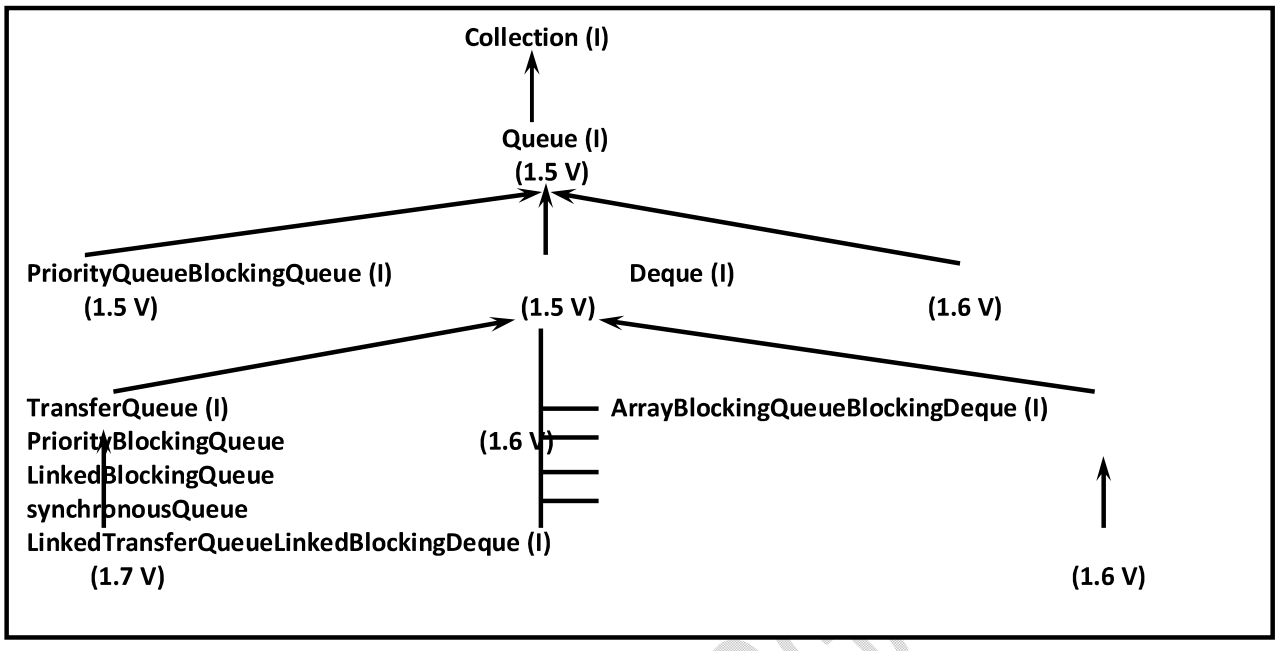


BlockingDeque (I) 1.6 V

- **It is the Child Interface of BlockingQueue and Deque.**
- **It is a Simple Deque with Blocking Operations but wait for the Deque to become Non Empty fro Retrieval Operation and wait for Space to Store Element.**



JAVA Means DURGASOFT


Collection (I)
Queue (I)
(1.5 V)
PriorityQueueBlockingQueue (I)
(1.5 V)
Deque (I)
(1.5 V)
(1.6 V)
TransferQueue (I)
PriorityBlockingQueue
LinkedBlockingQueue
synchronousQueue
LinkedTransferQueueLinkedBlockingDeque (I)
(1.7 V)
(1.6 V)
ArrayBlockingQueueBlockingDeque (I)
(1.6 V)


CORE JAVA with
OCJP/SCJP
JAVA CERTIFICATION
Mr. DURGA M.Tech
JAVA EXPERT
Trained Thousands of Students
SATISFACTION
100%
GUARANTEED
One to One
VIDEO
CLASSES
EVERYTHING AT YOUR CONVENIENCE
At your convenient Time
With in your convenient duration
AN ISO 9001:2008 CERTIFIED
DURGA
Software Solutions®
www.durgasoft.com
202, 2nd Floor, HUDA Maitrivanam,
Ameerpet, Hyd. Ph: 040-64512786,
9246212143, 8096969696


LEARN FROM EXPERTS ...
COMPLETE JAVA
CORE JAVA, ADV. JAVA, ORACLE, STRUTS, HIBERNATE, SPRING, WEB SERVICES,...
COMPLETE .NET
C#.NET, ASP.NET, SQL SERVER, MVC 5 & WCF
TESTING TOOLS
MANUAL + SELENIUM
ORACLE
D2K
MSBI
SHARE POINT
HADOOP
ANDROID
C, C++, DS, UNIX
CRT & APTITUDE TRAINING
AN ISO 9001:2008 CERTIFIED
DURGA
Software Solutions®
202, 2nd Floor, HUDA Maitrivanam,
Ameerpet, Hyd. Ph: 040-64512786,
9246212143, 8096969696
www.durgasoft.com