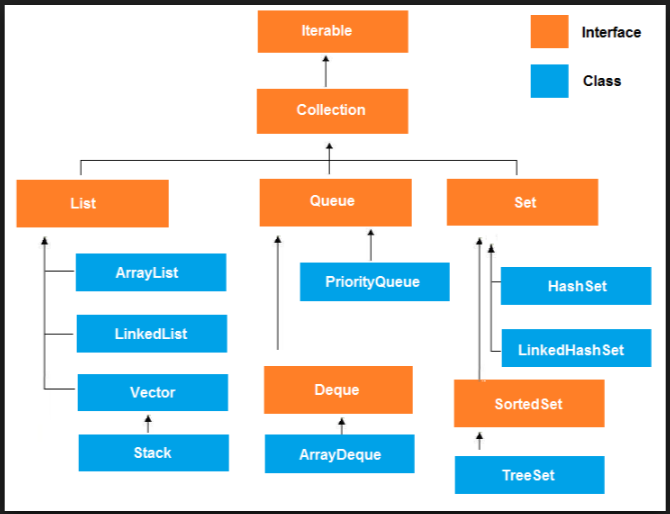
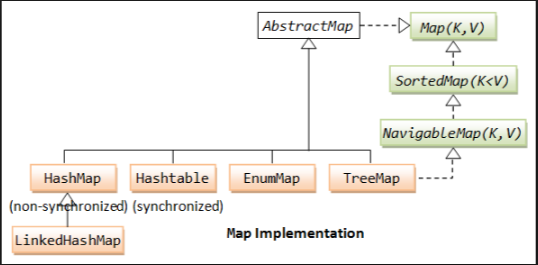
* **Collections Framework**

****

****

1. **What is Collection? What is a Collections Framework? What are the benefits of Java Collections Framework?**

**Collection**: A collection (also called as container) is an object  that groups multiple elements into a single unit.

**Collections Framework**: Collections framework provides unified architecture for manipulating and representing collections.

**Benefits of Collections Framework:**   
1. Improves program quality and speed  
2. Increases the chances of reusability of software  
3. Decreases programming effort.

1. **What is the root interface in collection hierarchy? IMP**Collection interface .

Few interviewers may argue that   
Collection interface extends Iterable interface. So iterable should be the root interface. But you should reply iterable interface present in java.lang package not in java.util package .It is clearly mentioned in [Oracle Collection  docs](http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html) , that Collection interface is a member of the Java Collections framework.  For [Iterable interface Oracle doc](https://docs.oracle.com/javase/7/docs/api/java/lang/Iterable.html" \t "_blank) , iterable interface is not mentioned as a part of the Java Collections framework .So if the question includes  collection hierarchy , then you should answer the question as Collection interface (which is found in java.util package)

1. **What is the difference between Collection and Collections?**Collection is an interface while Collections is a java class; both are present in java.util package and part of java collections framework.
2. **Which collection classes are synchronized or thread-safe?**Stack, Properties, Vector and Hash table can be used in multi threaded environment because they are synchronized classes (or thread-safe).
3. **How do you convert a given Collection to Synchronized Collection? IMP**  
   One line code:    Collections.synchronizedCollection(Collection collectionObj) will convert a given collection to synchronized collection.
4. **How will you make Collections readOnly ? IMP**

We can make the Collection readOnly by using the following lines code:

General : Collections.unmodifiableCollection(Collection c)  
Collections.unmodifiableMap(Map m)

Collections.unmodifiableList(List l)

Collections.unmodifiableSet(Set s)

1. **What is UnsupportedOperationException?**  
   This exception is thrown to indicate that the requested operation is not supported.  
   Example of UnsupportedOperationException:  
   In other words, if you call add() or remove() method on the readOnly collection . We know readOnly collection cannot be modified. Hence, UnsupportedOperationException will be thrown.
2. **Name the core Collection interfaces? IMP**

The list of core collection interfaces are : just mention the important ones  
Important : Collection , List, Queue ,Set Map  
Other interface also in the list :  Deque, SortedSet, SortedMap , NavigabelMap, ListIterator etc

-------------------------------------------------------------------------------------------------------------------------------------

* **List**

1. **Characteristics of ArrayList? IMP**

Tip: Any collection characteristics you should remembered based on points.

A. Insertion of records

B. storing of those records (will it store duplicate?)

C. Reading of that record.   
D. deleting that record.

1. Maintains insertion order
2. Can have duplicate elements
3. Array works at indexed basis as a result it allows random access
4. **How Array List works internally? IMP Better to learn by writing the complete class itself**

Array List grows internally.

When a constructor is called an array of size 10 will be called created by default. We can also pass the size.

ADD

When we try to add an element first add method checks whether internally created array of objects has enough space to accommodate the new element. If it cannot accommodate the new element (current array list size +1 > array objects.size )then existing array list grows internally. Arraylist is growing means a new arraylist of new size will be created and the existing objects will be copied to the object.

Formula to calculate new Arraylist size:

Till Java 6: around 50% of existing list

int newCapacity = (oldCapacity \* 3)/2 + 1;

(Update) From Java 7

int newCapacity = oldCapacity + (oldCapacity >> 1);

Can read more here:

If required need to learn remove also

http://javahungry.blogspot.com/2015/05/how-add-method-works-internally-in-arraylist.html

1. **What is the formula to resize the arrayList? IMP**

Formula to calculate new Arraylist size:

Till Java 6: around 50% of existing list

int newCapacity = (oldCapacity \* 3)/2 + 1;

(Update) From Java 7

int newCapacity = oldCapacity + (oldCapacity >> 1);

1. **Two important questions about ArrayList remove? IMP**

http://www.java67.com/2014/03/2-ways-to-remove-elementsobjects-from-ArrayList-java.html

1. **What are the classes implementing List interface?**

ArrayList , Vector , LinkedList

1. **How to reverse the List in Collections?**   
   There is a built in reverse method in Collections class . reverse (List list) accepts list as parameter.  
   Collections.reverse(listobject);
2. **How to convert the array of strings into the list?**Arrays class of java.util package contains the method asList() which accepts the array as parameter.  
   So,  
   String[]  wordArray =  {"Love Yourself"  , "Alive is Awesome" , "Be in present"};  
   List wordList =  Arrays.asList(wordArray);
3. **What is the default size of ArrayList? IMP**

10

1. **Doubly linked list class implementation? IMP little low priority but better to know**
2. **Difference Between Vector and Arraylist ? (I)**
3. **Synchronization and Thread-Safe**: Vector is synchronized while ArrayList is not synchronized.
4. **Performance**: Vector is slow as it is thread safe . In comparison ArrayList is fast as it is non synchronized
5. **Automatic Increase in Capacity**: A Vector defaults to doubling size of its array . While when you insert an element into the ArrayList , it increases its Array size by 50% .

By default ArrayList size is 10. It checks whether it reaches the last element then it will create the new array ,copy the new data of last array to new array ,then old array is garbage collected by the Java Virtual Machine (JVM) .

1. **Enumerator**: Other than Hashtable ,Vector is the only other class which uses both Enumeration and Iterator .While ArrayList can only use Iterator for traversing an ArrayList .
2. **Set increment size**: ArrayList does not define the increment size . Vector defines the increment size .You can find the following method in Vector Class

Public synchronized void setSize(int i) { //some code }

There is no setSize() method or any other method in ArrayList which can manually set the increment size.

1. **Difference between Array and Array List in Java with Example? (I)**
2. **Resizable:** Array is static in size that is fixed length data structure, One can not change the length after creating the Array object.

Array List is dynamic in size. Each ArrayList object has instance variable capacity which indicates the size of the ArrayList. As elements are added to an ArrayList its capacity grows automatically.

1. **Primitives:** ArrayList cannot contains primitive data types (like int , float , double) it can only contains Object while Array can contain both primitive data types as well as objects.

One get a misconception that we can store primitives(int,float,double) in ArrayList , but it is not true

Suppose we have ArrayList object,

ArrayList arraylistobject = new ArrayList();

arraylistobject.add(23); // try to add 23 (primitive)

JVM through Autoboxing(converting primitives to equivalent objects internally) ensures that only objects are added to the arraylist object.

thus , above step internally works like this :

arraylistobject.add( new Integer(23));

// Converted int primitive to Integer object and added to arraylistobject

1. **Multi-dimensional**

Array can be multi dimensional, while ArrayList is always single dimensional.  
example of multidimensional array:  
Integer addarrayobject[][] = new Integer[3][2];  
addarrayobject[0][0]= new Integer(8)

Note: more here : [**http://javahungry.blogspot.com/2015/03/difference-between-array-and-arraylist-in-java-example.html**](http://javahungry.blogspot.com/2015/03/difference-between-array-and-arraylist-in-java-example.html)

1. **Write the code for iterating the list in different ways in java?  (I)**

There are two ways to iterate over the list in java:   
a. using Iterator  
b. using for-each loop

c. you can use java8 forEach(Consumer)  
Coding part : Do it  yourself (DIY) ,

1. **What is CopyOnWriteArrayList ?  How it is different from ArrayList in Java?**  
   [CopyOnWriteArrayList](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/CopyOnWriteArrayList.html) is a thread safe variant of ArrayList   in which all mutative operations like add , set are implemented by creating a fresh copy of the underlying array.  
   It guaranteed not to throw ConcurrentModificationException.  
   It permits all elements including null. It is introduced in jdk 1.5.

------------------------------------------------------------------------------------------------------------------------------------

* **Queue**

1. **What is queue? IMP If time permits you can see implementation of some of collection queues**

It represents an ordered list of objects

In queue elements inserted at the end of the queue, and elements removed from the beginning of the queue

1. **What is deque? IMP**

Queue that supports insertion and removal of elements at both end points.

1. **What is the difference between poll (), remove() and peek () method of the Queue interface ?**

All three are remove related methods.

**Poll** (): Retrieves and removes the head of this queue, or returns null if this queue is empty. **Remove** (): Retrieves and removes the head of this queue but it throws a NoSuchElementException if this queue is empty.

**Peek** () Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.

1. **What is BlockingQueue in Java Collections Framework? IMP**  
   [Blocking Queue](https://docs.oracle.com/javase/6/docs/api/java/util/concurrent/BlockingQueue.html) implements the java.util.Queue interface . BlockingQueue supports  operations that wait for the queue to become non-empty when retrieving an element , and wait  for space to become available in the queue when storing an element .  
   It does not accept null elements.  
   Blocking queues are primarily designed for the producer-consumer problems.  
   Blocking Queue implementations are thread-safe and can also be used in inter-thread communications

Example for this is writing log in asynchronous way. We have implemented that in consumer cloud RAS while writing interface log.

Also in pre-processor we were reading data from Kafka and processing was taking lot of time so we implemented blocking queue and one thread was reading and storing and multiple threads were processing

------------------------------------------------------------------------------------------------------------------------------------

* **Set**

1. **Set characteristics?**

Set will not maintain order in which records are inserted.

Set contains **only unique elements** as it used map internally

1. **How hashSet works internally or how uniqueness is maintained in hash set? IMP**

Hash set internally uses hash map. So when hash set object is created internally hashmap object will be initialized. So when hashSet.put (E e) is called, internally hashMap.put(e,PRESENT(dummy object)) will be invoked key the object passed to hashSet is key and PRESENT is just a dummy object (Object PRESENT = new Object());.

So we know that in hashMap keys will be always unique. So objects inserted in to set are also unique. Put code looks like below.

public boolean add(E e) {

return map.put(e, PRESENT)==null;

}

We know that if key value pair passed to put is inserted then null will be returned by put method as a result add returns true when the element is added successfully.

And if duplicated key is passed to put then value of the key will be replaced and old value will be returned, in this case if already existing element is passed then PRESENT object will be returned back as a result condition is evaluated to false and same is returned which indicated object is duplicate.

<http://javahungry.blogspot.com/2013/08/how-sets-are-implemented-internally-in.html>

1. **What are the classes implementing Set interface? (B)**

HashSet, TreeSet

1. **Difference between HashSet and TreeSet  ? (I)**
2. **Ordering:** HashSet stores the object in random order. Elements are sorted according to the natural ordering of its elements in TreeSet. If the objects cannot be sorted in natural order than use compareTo() method to sort the elements of TreeSet object.
3. **Null value:** HashSet can store null object while TreeSet does not allow null object. If one try to store null object in TreeSet object , it will throw Null Pointer Exception.
4. **Speed:** HashSet is much faster than TreeSet,as performance time of HashSet is constant against the log time of TreeSet for most operations (add,remove ,contains and size) . Iteration performance of HashSet mainly depends on the load factor and initial capacity parameters.
5. **Internal implementation:** HashSet are internally backed by hashmap. While TreeSet is backed by a Navigable Tree Map.
6. **Comparision**: HashSet uses equals() method for comparison in java while TreeSet uses compareTo() method for maintaining ordering .
7. **What copy technique internally used by HashSet clone() method ?**  
     
   There are two copy techniques in every object oriented programming lanuage , deep copy and shallow copy.  
     
   To create a clone or copy of the Set object, HashSet  internally uses shallow copy in clone() method , the elements themselves are not cloned . In other words , a shallow copy is made by copying the reference of the object.
8. **Why HashSet does not have get (Object o) method?**

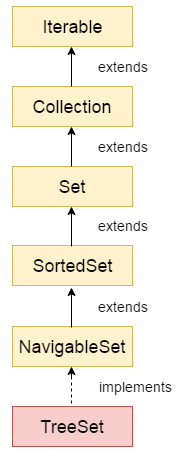
get(Object o) is useful when we have one information linked to other information just like key value pair found in HashMap .So using get() method on one information we can get the second information or vice-versa.  
  
Unlike HashMap , HashSet is all about having unique values or unique objects . There is no concept of keys in HashSet .  
The only information we can derive from the HashSet object is whether the element is present in the HashSet Object or not . If the element is not present in the HashSet then add it otherwise return true leaving HashSet object unchanged. Here, contains() method helps to provide this information.  
  
Due to the above reason there is no get(Object o) method in HashSet.

More details on HashSet Refer: <http://javahungry.blogspot.com/2014/04/top-10-hashset-java-interview-questions-collection.html>

1. **TreeSet Characteristics?**

Contains unique elements

Maintains ascending order



1. **What is EnumSet in Java? Need to study**

[EnumSet](http://docs.oracle.com/javase/7/docs/api/java/util/EnumSet.html)  is a specialized Set implementation for use with enum types. All of the elements in an enum set must come from a single enum type that is specified explicitly  or implicitly , when the set is created.  
The iterator never throws ConcurrentModificationException and is weakly consistent.  
Advantage over HashSet:  
All basic operations of EnumSet execute in constant time . It is most likely to be much faster than HashSet counterparts.  
It is a part of Java Collections Framework since jdk 1.5

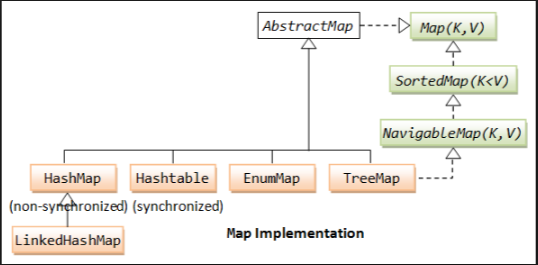
---------------------------------------------------------------------------------------------------------------------------

1. **Difference between ArrayList and LinkedList in Java**? **(I)**
2. **Implementation:**ArrayList is the resizable array implementation of list interface, while LinkedList is the Doubly-linked list implementation of the list interface.
3. **Reverse Iterator :**LinkedList can be iterated in reverse direction using descendingIterator() while there is no descendingIterator() in ArrayList , so we need to write our own code to iterate over the ArrayList in reverse direction.
4. **Memory Overhead:**Memory overhead in LinkedList is more as compared to ArrayList as node in LinkedList needs to maintain the addresses of next and previous node. While in ArrayList  each index only holds the actual object(data).
5. **Initial Capacity :** If the constructor  is not overloaded , then ArrayList creates an empty list of initial capacity 10 , while LinkedList  only constructs the empty list without any initial capacity.
6. **When to use ArrayList and when to use LinkedList in application? (I)**

ArrayList has constant time search operation O(1) .Hence, ArrayList is preferred when there are more get() or search operation .  
Insertion , Deletion operations take constant time O(1) for LinkedList. Hence, LinkedList is preferred when there are more insertions or deletions involved in the application.

1. **What is the difference between List and Set? (B)**  
     
   **Duplication:** List can contain duplicate elements , while Set contain only unique elements.  
   **Order:** List is ordered while Set is unordered. List maintains the order in which the objects are added.
2. **What is the difference between Map and Set? (B)**  
   Map object has unique keys each containing some value, while Set contain only unique values.
3. **What is the difference between Queue and Stack? (B)**  
   Queue is a data structure which is based on FIFO ( first in first out ) property . An example of Queue in real world is buying movie tickets in the multiplex or cinema theaters.  
     
   Stack is a data structure which is based on LIFO (last in first out) property . An example of Stack in real world is  insertion or removal of CD  from the CD case.

* **Map**

****

**HashMap**

1. **How Hash Map Works In Java? IMP**

HashMap works on the principle of **Hashing**.

**Hashing**: Hashing is technique where “*we can convert String or object to short fixed length Integer. This helps in faster indexing and look-up.”*

Before going further we should know three terms

1. **Hash Function**: The function which converts object/String to fixed integer value is called as hash function.
2. **Hash Value:** value returned by hash function is called as has value
3. **Bucket**: Bucket is used store key value pair. A bucket can have multiple key value pair. Bucket uses simple linked list to store key value pair.

When HashMap object is created by default array of Entry objects of size 16 will be created here. This Entry object array is called as bucket.

Entry is inner class of Map Entry object will have **key, value, hash, and Entry** (reference to another entry for linked list).

**Put (K, V).**

When put method is invoked with key and value.

1. **If key is null**: then it will always be inserted at index zero coz hash code of null is always zero.
2. **If key is not null:** First of all, hash code of key will be computed and the returned hash code will be hashed again by the internal hash method to defend against poor hash code implementations.
3. Using the hash code generated and length of the bucket index at which key value pair to be stored will e calculated using bitwise &. (hash & (length - 1)).
   1. If there is no Entry object at the index calculated. The newly created Entry object for input key, value and hash will be pointed to the calculated index with Entry parameter as null.
   2. If there is an entry object already pointing to index calculated, newly created object will be inserted at the front to form linked list. New object will be inserted at the front in the linked list. (That is existing entry object will be pointed to newly created entry object’s entry reference and newly created entry object will be pointed to corresponding index of bucket.)
   3. If the input key is already present in the bucket then its value will be replaced.

**get (K):**

When get is called

1. **With null key :**  As it is already known fact that hash code of null is 0 . Key will be searched at index zero and if any key is matched with null corresponding value will be returned.
2. **If key is not null**: hash code of the key will be calculated and it will be further hashed and index will be calculated by using **hashcode & (length-1).** Once index is found all the Entry objects at that index will be looped (entry objects are stored as linked list form) by comparing hash and key once the match is found the corresponding value will be returned. If match is not found then null will be returned.

**remove(Key)**

As we know to find the desired Entry object which is to be removed in the HashMap we need hashValue , key and bucketindex . So remove(key) method calls  removeEntryForKey(key) method  internally , which calculate the final hashValue of the key object , and then use that hashValue in the indexFor(int,int) method to find the first entry object in the appropriate bucket.   
Since bucket(table) is a LinkedList effectively , we start traversing from the first entry object which we got by using indexFor(int,int) method in the bucket. For each entry object in the bucket we compare whether  hashValue and the key is equal to the calculated hashValue in the first step and the key passed as a parameter in the remove(key) method.  
If desired Entry object is found , then we removed that single entry object from the LinkedList.   
Removing a single Entry object from the LinkedList is implemented just like removing a single object from the LinkedList.  
  
Entry object returned by the removeEntryForKey(key) method is then  stored in the local variable e of type Entry in the remove(key) method.  
  
If (e==null)  
     return null  
else  
    return value of removed Entry object.

1. **How will you measure the performance of Hash Map? IMP**  
   Performance of the Hash map is mainly depending on two factors:

**Capacity and load factor**.

**Capacity**: The capacity is the number of buckets in the hash table and the initial capacity is simply the capacity at the time the hash table is created.

**Load factor**: How full the hash map is allowed before its capacity is automatically increased.

Threshold value in the Hash map is calculated using **capacity \* load factor.**

If number of entries in hash map exceeds the threshold value hash map will be resized to double the current size.

1. **What is default capacity and default load factor for hash map?**  
   **Capacity**: 16 **capacity of bucket should be in power of 2**

**Load factor:** 0.75f

1. **What is the time complexity of Hashmap get() and put() method ?** **IMP**  
   The hashmap implementation provides constant time performance for (get and put) basic operations  
   i.e the complexity of get () and put () is **O (1)**, assuming the hash function disperses the elements properly among the buckets.

Ref: <https://www.youtube.com/watch?v=c3RVW3KGIIE>

<http://javahungry.blogspot.com/2013/08/hashing-how-hash-map-works-in-java-or.html>

1. **How do you use a custom object as key in Collection classes like Hash Map? IMP**

By override equals() and hashCode() method.

1. **What is the use of hashcode in java? IMP**
2. **In which class hashCode method is there? IMP**

Object

1. **What is the contract between equals and hash code? IMP**  
   1. If two objects are equal, then they must have the same hash code.

2. If two objects have the same hash code, they may or may not be equal.

 If  object1.equals(object2) , then  object1.hashCode() == object2.hashCode() should always be true.

If object1.hashCode() == object2.hashCode() is true does not guarantee object1.equals(object2)

if (e.hash == hash && ((k = e.key) == key || key.equals(k)))

1. **When we use iterator to loop hashmap can we delete any record? IMP**

Iterator<Object> it = map.keySet().iterator();

while (it.hasNext())

{

it.next();

if (something)

it.remove();

}

<http://stackoverflow.com/questions/1884889/iterating-over-and-removing-from-a-map>

1. **Inside Entry what are the classes defined? IMP If you can remember the code it is well and good**
2. **What is LinkedHashMap**?

LikedHashMap it is same as HashMap but *maintains insertion order.* It means that when you iterate through LinkedHashMap you will get the value in the same way you inserted.

**HashTable**

1. **What is hash-collision in Hash table? How it was handled in Java? IMP**  
   if two different keys have the same hash value then it lead to hash -collision.

A bucket of type LinkedList used to hold the different keys of same hash value.

1. **Difference between HashMap and HashTable? IMP**
2. **Synchronization or Thread Safe:** HashMap is non synchronized and not thread safe. On the other hand, HashTable is thread safe and synchronized.

When to use HashMap ? Answer is if your application do not require any multi-threading task, in other words hashmap is better for non-threading applications. HashTable should be used in multithreading applications.

1. **Performance**: Hashmap is much faster and uses less memory than Hashtable as former is unsynchronized. Unsynchronized objects are often much better in performance in compare to synchronized object like Hashtable in single threaded environment.
2. **Null keys and null values:** Hashmap allows one null key and any number of null values, while Hashtable do not allow null keys and null values in the HashTable object
3. **Iterating the values:**  Hashmap object values are iterated by using iterator .HashTable is the only class other than vector which uses enumerator to iterate the values of HashTable object.
4. **Fail-fast iterator:** The iterator in Hashmap is fail-fast iterator while the enumerator for Hashtable is not.

According to Oracle Docs, if the Hashtable is structurally modified at any time after the iterator is created in any way except the iterator's own remove method , then the iterator will throw ConcurrentModification Exception.

Structural modification means adding or removing elements from the Collection object (here hashmap or hashtable) . Thus the enumerations returned by the Hashtable keys and elements methods are not fail fast.

1. **Superclass and Legacy**: Hashtable is a subclass of Dictionary class which is now obsolete in Jdk 1.7, so, it is not used anymore. It is better off externally synchronizing a HashMap or using a ConcurrentMap implementation (e.g ConcurrentHashMap).HashMap is the subclass of the AbstractMap class. Although Hashtable and HashMap has different superclasses but they both are implementations of the "Map” abstract data type.
2. **What is the purpose of calling recordRemoval() method in the removeEntryForKey(key)  since it is the concrete method without anybody?**

My understanding: The method removeEntryForKey(key) is used by both hashmap and linked haskmap. So the method recordRemoval() is required for linked hashmap to remove the record but in case of hashmap entry will be removed by removing its reference from hashmap bucket. So hashMap will be having concrete method without any body.

WebSite: recordRemoval() method is a concrete method without any body. It is invoked whenever the Entry is removed from the table . Since LinkedHashMap extends HashMap , thus this method is overridden in the LinkedHashMap's Entry in order to maintain its linked list of entries.

Refer below link for additional questions: it will be better if you can write remove methods code

<http://javahungry.blogspot.com/2015/03/how-remove-method-internally-works-in-hashmap-java.html>

1. **Write java code showing insertion,deletion and retrieval of HashMap object ? (I)**

WRITE CODE LATER.

Segments's constructor calls ReentrantLock's no arg constructor and creates a non fair lock.

1. **HashMap vs ConcurrentHashMap? IMP**

|  |  |  |
| --- | --- | --- |
| **Category** | **Hashmap** | **Concurrent hash map** |
| **Thread –Safe** | not thread-safe. | thread-safe |
| **Synchronization** | HashMap can be synchronized by using  synchronizedMap(HashMap) method .  By using this method, we get a HashMap object which is equivalent to the Hash Table object. So every modification is performed on Map is locked on Map object | ConcurrentHashMap synchronizes or locks on at bucket level. So that other threads can access other elements of the bucket |
| **Performance** | Faster as it is not synchronized | Slower compare to hashmap because of bucket level synchronization |
| **Null key** | Hash map can have one null key and multiple null values | Concurrent hashmap does not allow null keys and values |

1. **If I synchronize the hash map, then will it work similar to concurrent hash map? IMP**
2. **How Concurrent HashMap works internally? IMP**

<https://www.geeksforgeeks.org/concurrenthashmap-in-java/>

The underlined data structure for ConcurrentHashMap is Hashtable.

ConcurrentHashMap internally divides buckets into segments and one particular segment can be locked by a thread for writing purpose.

Segment is nothing but a static class which is specialized version of hash tables and implements Reentrantlock to simplify locking as

static class Segment<K,V> extends ReentrantLock implements Serializable {

private static final long serialVersionUID = 2249069246763182397L;

final float loadFactor;

Segment(float lf) { this.loadFactor = lf; }

}

1. **What is Reentrant lock in java? IMP**

[**https://www.geeksforgeeks.org/reentrant-lock-java/**](https://www.geeksforgeeks.org/reentrant-lock-java/)

The traditional way to achieve thread synchronization in Java is by the use of synchronized keyword. While it provides a certain basic synchronization, the synchronized keyword is quite rigid in its use. For example, a thread can take a lock only once. Synchronized blocks don’t offer any mechanism of a waiting queue and after the exit of one thread, any thread can take the lock. This could lead to starvation of resources for some other thread for a very long period of time.

Reentrant Locks are provided in Java to provide synchronization with greater flexibility.

What are Reentrant Locks?

The ReentrantLock class implements the Lock interface and provides synchronization to methods while accessing shared resources. The code which manipulates the shared resource is surrounded by calls to lock and unlock method. This gives a lock to the current working thread and blocks all other threads which are trying to take a lock on the shared resource.

As the name says, ReentrantLock allows threads to enter into the lock on a resource more than once. When the thread first enters into the lock, a hold count is set to one. Before unlocking the thread can re-enter into lock again and every time hold count is incremented by one. For every unlocks request, hold count is decremented by one and when hold count is 0, the resource is unlocked.

Reentrant Locks also offer a fairness parameter, by which the lock would abide by the order of the lock request i.e. after a thread unlocks the resource, the lock would go to the thread which has been waiting for the longest time. This fairness mode is set up by passing true to the constructor of the lock.

These locks are used in the following way:

1. **Arrange the following in the ascending order (performance):  
   HashMap , Hashtable , ConcurrentHashMap and Collections.SynchronizedMap?**

Hashtable < Collections.SynchronizedMap < ConcurrentHashMap < HashMap

1. **Why Map interface does not extend the Collection interface in Java Collections Framework?**

One liner answer : Map interface is not compatible with the Collection interface.  
Explanation : Since Map requires key as well as value , for example , if we want to add key-value pair then we will use put(Object key , Object value) . So there are two parameters required to add element to the HashMap object  . In Collection interface add(Object o) has only one parameter.   
The other reasons are Map supports values set , keySet as well as other appropriate methods which have just different views from the Collection interface.

**TreeMap**:

1. **What are characteristics of TreeMap? IMP**

Tree map is used to store key/value pairs in sorted order (ascending order).

It *can not* have null key. Can have multiple null values

1. **What is the difference between HashMap and TreeMap? IMP**

HashMap can contain one null key.

TreeMap can not contain any null key.

HashMap maintains no order.

TreeMap maintains ascending order.

1. **When do i need tree map? IMP**
2. **How Tree Map works in Java?**

**Yet to study.**

[**http://javahungry.blogspot.com/2014/06/how-treemap-works-ten-treemap-java-interview-questions.html**](http://javahungry.blogspot.com/2014/06/how-treemap-works-ten-treemap-java-interview-questions.html)

1. **IdentityHashMap?**

It implements the Map interface with a hash table , using [**reference equality instead of object equality**](http://javahungry.blogspot.co.uk/2013/06/difference-between-equals-and-double-equals-method-with-example-java-collections-interview-question.html)**when comparing keys and values**.In other words , in IdentityHashMap two keys k1 and k2 are considered equal if only if (k1==k2).

IdentityHashMap is not synchronized.

Iterators returned by the iterator() method are fail-fast , hence , will throw ConcurrentModificationException.

1. **What is WeakHashMap** ?

It is a Hashtable based implementation of Map interface with weak keys. **An entry in WeakHashMap will automatically be removed when its key is no longer in ordinary use (Key also should have reference)**. More precisely the presence of a mapping for a given key will not prevent the key from being discarded by the garbage collector.

It permits null keys and null values.  
Like most collection classes this class is not synchronized.A synchronized WeakHashMap may be constructed using the Collections.synchronizedMap() method.

Iterators returned by the iterator() method are fail-fast , hence , will throw ConcurrentModificationException.

1. **What are concurrentCollectionClasses?**   
   In jdk1.5 , Java Api developers had introduced new package called java.util.concurrent that have thread-safe collection classes as they allow collections to be modified while iterating . The iterator is fail-safe that is it will not throw ConcurrentModificationException.  
   Some examples of concurrentCollectionClasses are :  
   a. CopyOnWriteArrayList  
   b. ConcurrentHashMap

* **Properties**

1. **Collection value of type properties is defined by?**

--------------------------------------------------------------------------------------------------------------------------------------

* **Iterator**

1. **What is an iterator ? (B)**  
   Iterator is an interface. It is found in java.util package. It provides methods to iterate over any Collection.
2. **Fail Safe Iterator:**   
   Fail Safe Iterator makes copy of the internal data structure (object array) and iterates over the copied data structure. Any structural modification done to the iterator affects the copied data structure.  So, original data structure remains structurally unchanged .Hence , no ConcurrentModificationException throws by the fail safe iterator.

Two issues associated with Fail Safe Iterator are:   
1. Overhead of maintaining the copied data structure i.e memory.  
2.  Fail safe iterator does not guarantee that the data being read is the data currently in the original data structure.

1. **What is the difference between Iterator and Enumeration ?**  **(B)**  
   The main difference between Iterator and Enumeration is that Iterator has remove() method while Enumeration doesn't.  
   Hence, using Iterator we can manipulate objects by adding and removing the objects from the collections. Enumeration behaves like a read only interface as it can only traverse the objects and fetch it.
2. **Which design pattern followed by Iterator ? (B)**  
   It follows iterator design pattern. Iterator design pattern provides us to navigate through the collection of objects by using a common interface without letting us know about the underlying implementation.  
   Enumeration is an example of Iterator design pattern.
3. **Fail fast Iterator IMP**

While iterating through the collection if there is structural modification of the collection instantly throws Concurrent Modification Exception.

Fail-fast iterator can throw ConcurrentModificationException in two scenarios:

Single Threaded Environment  
After the creation of the iterator , structure is modified at any time by any method other than iterator's own remove method.

Multiple Threaded Environment  
If one thread is modifying the structure of the collection while other thread is iterating over it.

**Recap: Difference between Fail Fast Iterator and Fail Safe Iterator IMP**

|  |  |  |
| --- | --- | --- |
|  | **Fail Fast Iterator** | **Fail Safe Iterator** |
| Throw ConcurrentModification Exception | Yes | No |
| Clone object | No | Yes |
| Memory Overhead | No | Yes |
| Examples | HashMap,Vector,ArrayList,HashSet | CopyOnWriteArrayList, ConcurrentHashMap |

1. **What is the difference between Iterator and ListIterator?**

Using Iterator we can traverse the list of objects in forward direction. But ListIterator can traverse the collection in both directions that is forward as well as backward.

* **Comparable and Comparator**

1. **Comparable and Comparator in Java? IMP**

Java provides some inbuilt methods to sort primitive types array or Wrapper classes array or list.

Using **java.lang.Comparable** and **java.util.Comparator** interfaces we can sort array/list of custom classes.

**Comparable**: Java provides **Comparable** interface which should be implemented by any custom class if we want to use Arrays or Collections sorting methods. Comparable interface has **compareTo(T obj)** method which is used by sorting methods, you can check any Wrapper, String or Date class to confirm this. We should override this method in such a way that it returns a negative integer, zero, or a positive integer if “this” object is less than, equal to, or greater than the object passed as argument.

Comparable interface can sort custom objects based on one parameter. It does not support multiple parameters but Comparator interface will do that.

**Comparator**: Comparator interface *compare(Object o1, Object o2)* method need to be implemented that takes two Object argument, it should be implemented in such a way that it returns negative int if first argument is less than the second one and returns zero if they are equal and positive int if first argument is greater than second one.

So now we know that if we want to sort java object array or list, we need to implement java Comparable interface to provide default sorting and we should implement java Comparator interface to provide different ways of sorting.

## Comparable vs Comparator

1. **What is Comparable interface?**

**https://www.geeksforgeeks.org/comparable-vs-comparator-in-java/**

**java.lang.Comparable<T>**

**public interface Comparable<T>**

**{**

**public int compareTo(T o);**

**}**

If any class implements comparable interface that class, its object will get capability to compare itself with other object.

We can use this interface for object sorting.

For example: Movie class that has members like, rating, name, year. Suppose we wish to sort a list of Movies based on year of release. We can implement the Comparable interface with the Movie class, and we override the method compareTo() of Comparable interface.

**Working of compareTo method.**

The compareTo() method works by returning an int value that is either positive, negative, or zero. A negative number means that the object making the call is “less” than the argument. 0(zero) means that both objects are equal. A positive number means that the object making the call is “greater” than the argument.

Using comparable objects, we can sort only based on one member variable of the object for example in the above example it is year. Suppose if we want to sort based on rating and name of the movie also it is not possible as we can implement compareTo method only once.

1. **What is Comparator interface.**

**java.util.Comparator<T>**

**public interface Comparator<T> {**

**int compare (T o1, T o2);**

**}**

Comparator is external to the object that we are comparing. It’s a separate class.

We create multiple separate classes (that implement Comparator) to compare by different members.

Collections class has a second sort () method and it takes Comparator. The sort () method invokes the compare () to sort objects.

1. Comparable interface can be used to provide single way of sorting whereas Comparator interface is used to provide different ways of sorting.
2. For using Comparable, Class needs to implement it whereas for using Comparator we don’t need to make any change in the class.
3. Comparable interface is in java.lang package whereas Comparator interface is present in java.utilpackage.
4. We don’t need to make any code changes at client side for using Comparable, Arrays.sort() or Collection.sort() methods automatically uses the compareTo() method of the class. For Comparator, client needs to provide the Comparator class to use in compare() method
5. **Suppose there is an Employee class. We add Employee class objects to the ArrayList. Mention the steps need to be taken, if I want to sort the objects in ArrayList using the employeeId attribute present in Employee class.**  
   a. Implement the Comparable interface for the Employee class and now to compare the objects by employeeId we will override the emp1.compareTo(emp2)  
   b. We will now call Collections class sort method and pass the list as argument, that is ,  
        Collections.sort(empList)

http://www.journaldev.com/780/comparable-and-comparator-in-java-example

List:

Supports zero based indexing

Vector

Synchronized

Dynamic array