<<interface>>

SortedMap

TreeMap

ConcurrentHashMap

HashMap

<<interface>>

Map

Properties

Hashtable

AbstractMap

Dictionary

ArrayDeque

LinkedList

<<interface>>

Deque

LinkedHashSet

Vector

TreeSet

HashSet

<<interface>>

SortedSet

Stack

LinkedList

ArrayList

<<interface>>

Collection

<<interface>>

Map

<<interface>>

Queue

<<interface>>

Set

<<interface>>

List

PriorityQueue

1. **What is Collection Framework?**

Collection framework represents an architecture to store and manipulate a group of objects.

Unlike an array for which the length(size) if fixed at the time of instantiating the array, objects of collection framework can resize to hold any number of elements.

All the classes and interfaces of this framework are present in *java.util* package.

*java.util.concurrent* package also provide few collection related classes which support concurrency/multi-thread access with better performance compared to collection classes present in java.util package.

In java.util we have classes related to Traditional Collections, and in java.util.concurrent package we have classes related to Concurrent Collections (thread safe collections).

1. **Important interfaces of Collections Framework (java.util package)**

*java.lang.Iterable* interface is the root interface for all collection classes, it has one abstract method iterator()

Collection interface extends the Iterable interface.

*List, Set* and *Queue* interfaces extend Collection.

*Deque* in an interface that extends Queue

*Map* is another interface that allows processing of Key-Value pairs. Map does not extend the Collection or Iterable interface but is still part of collections framework.

1. **What is Collections class?**

*Collections* is a utility class present in *java.util* package that we can use while using collections like List, Set, and Map etc.Some commonly used methods are

sort(),

copy()

[synchronizedCollection](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Collections.html#synchronizedCollection(java.util.Collection))​([Collection](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Collection.html)<T>  c)

[synchronizedList](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Collections.html#synchronizedList(java.util.List))​([List](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/List.html)<T> list)

[synchronizedMap](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Collections.html#synchronizedMap(java.util.Map))​([Map](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Map.html)<K,​V> m)

[synchronizedSet](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Collections.html#synchronizedSet(java.util.Set))​([Set](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Set.html)<T> s)

[unmodifiableCollection](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Collections.html#unmodifiableCollection(java.util.Collection))​([Collection](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Collection.html)<? extends T> c)

[unmodifiableList](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Collections.html#unmodifiableList(java.util.List))​([List](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/List.html)<? extends T> list)

[unmodifiableMap](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Collections.html#unmodifiableMap(java.util.Map))​([Map](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Map.html)<? extends K,​? extends V> m)

[unmodifiableSet](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Collections.html#unmodifiableSet(java.util.Set))​([Set](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Set.html)<? extends T> s)

NOTE: A *utility* class is a class in which all methods are public and static.

1. **Difference between *java.util* and *java.util.concurrent* packages.**

*java.util package:*

This package mainly contains interfaces and classes **traditional** collection framework and other utility classes like Collections, Arrays, Objects, Scanner, Random, StringTokenizer etc.

*[NOTE: Have some basic idea of Collections, Arrays, Objects, Scanner, Random, StringTokenizer classes. It will be very helpful in job interviews and project onboarding interviews]*

*java.util.concurrent package:*

This package mainly contains interfaces and classes that work on top of threads (Thread & Runnable) and collections framework classes from java.util package with inbuilt implementation for handling concurrency and parallelism.

Examples of classes from java.util.concurrent related to multi-threading are:

1. Executor
2. CountDownLatch
3. CyclicBarrier
4. Etc.

(<https://www.geeksforgeeks.org/java-util-concurrent-package/>)

* [ Notes related to Thread utilities will be covered in another document. This document is primarily to discuss collections.]

Examples of classes from java.util.concurrent related to collections are:

1. ConcurrentHashMap
2. ImmutableList
3. ImmutableSet
4. CopyOnWriteArrayList
5. CopyOnWriteArraySet
6. Etc.
7. **What does O(1) and O(n) imply in Big-O notation?**

O(1) is constant-time, which means that the operation does not depend on the input size

O(n) is linear time, which means that the operation changes linearly with input size.

1. **What is an ArrayList? / Explain how an ArrayList works?**
2. ArrayList is a resizable and array-based implementation of List Interface.
3. As ArrayList internally uses an array to store data contiguous block of memory is allocated to hold its elements. As it is a contiguous block, all the elements address is already known, hence ArrayList allows random access to elements.
4. Some points about ArrayList class:
   1. ArrayList class maintains insertion order
   2. ArrayList class can contain duplicate elements
   3. ArrayList class allows random access to its elements as it works on index basis
   4. ArrayList class is not synchronized
   5. You can add any number of null elements in the ArrayList class
5. Time complexity of ArrayList’s get(), add(), remove() operations:

get(): constant time

add():

If you are using add(E element), then it will append element at the last position and will take O(1), provided that the arrayList is not fullotherwise it will create a new arrayList of one and a half times the size of previous arrayList and copy all arrayList elements to this new arrayList,making it O(n).

If you are using add(int index, E element), then depending on the index n number of elements will have to be shifted to the right. So time complexity will be O(n) depending upon the number of elements to be shifted to the right.

remove():

For remove method it is similar to add() method except that the elements will have to be shifted to the left.

1. **What is default size of ArrayList?**

10

Notice the below piece of code in ArrayList.class file in Eclipse:

A close-up of a white background

Description automatically generated

1. **Which data structure is used internally in an ArrayList?**

Internally, ArrayList uses an array of java.lang.Object class

A close-up of a computer screen

Description automatically generated

1. **How add() method works internally or How the ArrayList grows at runtime?**

add() method of ArrayList will add an element at the end of the array internally used by the ArrayList class.

It the current length of the internal array has been completely used, then a new array one-and-a-half times the length of current array is created, previously existing elements will be copies into the new array, the new element will be assigned to the first unused index in the array and the new array will be made the current array.

1. **How to make an ArrayList as Immutable?**

Use the static method *Collections.unmodifiableList()* in Collections class.

Any attempt to *add* or *remove* elements from the list object returned by *unmodifiableList*() method will result in UnsupportedOperationException.

*NOTE:* Declaring an ArrayList rererence variable with final keyword is not the correct answer. Doing so will not allow you to create and assign a new ArrayList object to the reference variable, but the elements inside the ArrayList will be modifiable.

1. **What is a LinkedList? / Explain how LinkedList works.**

Java LinkedList class is an implementation of List and it uses a doubly linked list to store the elements.

The LinkedList elements are not stored in contiguous locations, they are stored at any available space in memory and they are linked with each other using nodes.

As LinkedList class uses a double linked list, each node contains 3 parts:

1) Previous node reference

2) Data (the actual element)

3) Next node reference

Notice the below lines of code of LinkedList.class in Eclipse:

A screenshot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated

Some points about LinkedList class:

1. LinkedList class maintains insertion order
2. LinkedList class can contain duplicate elements
3. LinkedList class is not synchronized
4. LinkedList class implements the interfaces List, Stack and Queue
5. You can add any number of null elements in LinkedList

Time complexity of LinkedList’s get(), add() and remove():

**get():** As LinkedList does not store its elements in contiguous block of memory, random access is not supported here, elements can be accessed in sequential order only, so get() operation in LinkedList is O(n).

**add() and remove():** Both add and remove operations in LinkedList is O(1),because no elements shifting is needed, just pointer modification is done(although remember getting to the index where you want to add/remove will still be O(n))

1. **When to use ArrayList / LinkedList?**

When you have a requirement in which you will be doing a lot of add or remove operations near the middle of list, then prefer LinkedList, and when you have a requirement, where the frequently used operation is searching for an element from the list, then prefer ArrayList as it allows random access in an efficient way.

1. **What is HashMap? / Explain how HashMap works.**

HashMap class implements the Map interface and it stores data in key, value pairs. HashMap provides constant time performance for its get()and put() operations, assuming the equals and hashcode method has been implemented properly, so that elements can be distributed correctly among the buckets.

Some important points about HashMap:

1. Keys should be unique in HashMap, if you try to insert the duplicate key, then it will override the corresponding key’s value.
2. HashMap may have one null key and multiple null values
3. HashMap does not guarantee the insertion order (if you want to maintain the insertion order, use LinkedHashMap class)
4. Hashmap has a default initial capacity of 16, which means it has 16 buckets or bins to store map entries, each bucket is a singly linkedlist.
5. The default load factor in HashMap is 0.75-Load factor is that threshold value which when crossed will double the hashmap’s capacity i.e. when you add 13th element in hashmap, the capacity will increase from 16 to 32.
6. **Explain the internal working of put() and get() methods of HashMap class.**

put() method internal working:

When you call map.put(key,value), the below things happens:

1. Key’s hashCode() method is called.
2. Hashmap has an internal hash function which takes the key’s hashCode and it calculates the bucket index.
3. If there is no element present at that bucket index, our <key,value> pair along with hash is stored at that bucket.
4. But if there is an element present at the bucket index, then key’s hashCode is used to check whether this key is already present with the same hashCode or not. If there is key with same hashCode, then equals method is used on the key. If equals method returns true, then the key’s previous value is replaced with the new value otherwise a new entry is added to the bucket.

get() method internal working:

When you call map.get(key), the below things happen:

1. Key’s hashCode() method is called
2. Hash function uses this hashCode to calculate the index of the bucket.
3. Now the key of element stored in bucket is compared with the passed key using equals() method, if both are equal, value is returned otherwise the next element is checked. If no element in the bucket is found to be equal a null value is returned
4. **How to make a HashMap synchronized?**

Collections.synchronizedMap( *map* );

1. **What is ConcurrentHashMap?**

ConcurrentHashMap class provides concurrent access to the map.

To explain ConcurrentHashMap we have to compare it with HashMap and Hashtable.

- HashMap is not synchronized. Which means multiple threads can access a single HashMap object, hence thread-safety is an issue.

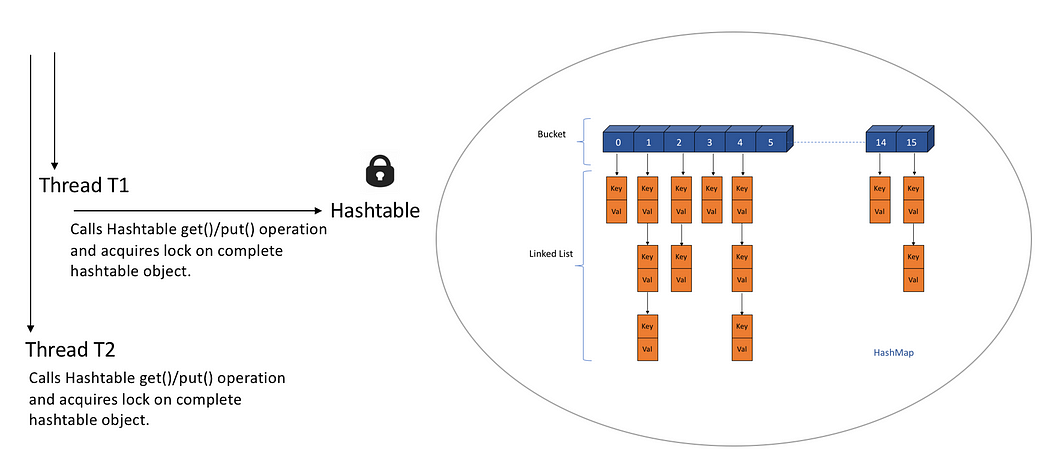
- Hashtable is synchronized. Which mean thread-safety is not an issue, but the the access is locked at the Hashtable object level. That is if one thread is accessing any feature of a Hashtable object, no operations (not even read operations) can be performed by other threads on the same Hashtable object. This leads to performance issues as other threads will have to wait for their chance.

ConcurrentHashMap is internally divided into segments. Unlike Hashtable, the entire map is not locked while reading/writing from ConcurrentHashMap.

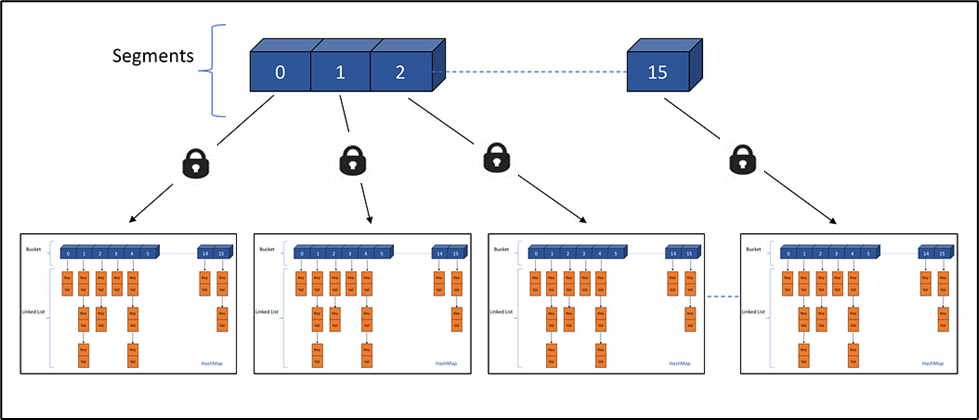
Lock is maintained at segment level. By default, 16 segments are maintained. Which means, maximum 16 threads can work at a time on a ConcurrentHashMap object.

The no.of segments can be changed at the time of instantiating the object of ConcurrentHashMap, as one of its constructors accepts a parameter for concurrency level.

Below image depicts the lock level with Hashtable. The entire object is locked by a single thread.



ConucrrentHashMap is further divided into SEGMENTS. Lock is maintained at segment level, so multiple threads can access the same ConcurrentHashMap object, but there will be a limit for the max no.of threads that can access. This is better than the object level lock.



1. **Explain java.util.TreeMap.**

In TreeMap key-value entries are sorted based on the natural ordering of its keys.

If we are using a custom class as the key, we have to make sure that the custom class is implementing Comparable interface.

TreeMap class also provides a constructor which takes a Comparator object, this should be used when we want to do a custom sorting.

1. **Explain java.util.TreeSet**

TreeSet class contains unique elements just like HashSet, but the elements are sorted in the natural order.

TreeSet class does not allow null elements.

TreeSet class internally uses TreeMap, i.e. the value added in TreeSet is internally stored in the key of TreeMap

If we are using a custom class as an element, we have to make sure that the custom class is implementing Comparable interface.

TreeSet class also provides a constructor which takes a Comparator object, this should be used when we want to do a custom sorting.

1. **Difference between fail-safe and fail-fast iterators**

Fail-fast iterators immediately throw ConcurrentModificationException, if the collection is modified while iterating over it.

Iterator of ArrayList and HashMap are fail-fast iterators.

Fail-safe iterators does not throw ConcurrentModificationException, because they operate on the clone of thecollection, not the actual collection.

Iterator of CopyOnWriteArrayList and ConcurrentHashMap are the examples of fail-safe iterators.

NOTE: Collection classes that provide fail-safe iterators are usually present in *java.util.****concurrent***package, while the collection classes that provide fail-fast iterators are present in java.util package.

1. **Difference between Iterator and ListIterator**

Iterator can traverse the collection only in one direction i.e.forward direction but ListIterator can traverse the list in bothdirections, forward as well as backward, using previous() andnext() method

Iterator cannot add element to a collection while iterating over it,but ListIterator can add elements while iterating over the list

Iterator cannot modify an element while iterating over acollection, but ListIterator has set(E e) method which can be usedto modify the element

Iterator can be used with List, Set or Map, but ListIterator onlyworks with List

1. **Difference between Iterator.remove and Collection.remove()**

Iterator.remove() does not throw ConcurrentModificationExceptionwhile iterating over a collection but Collection.remove() method will throwConcurrentModificationException