**Data Structure:**

Data structure is way of organizing a data in computer so that it can be used efficiently.

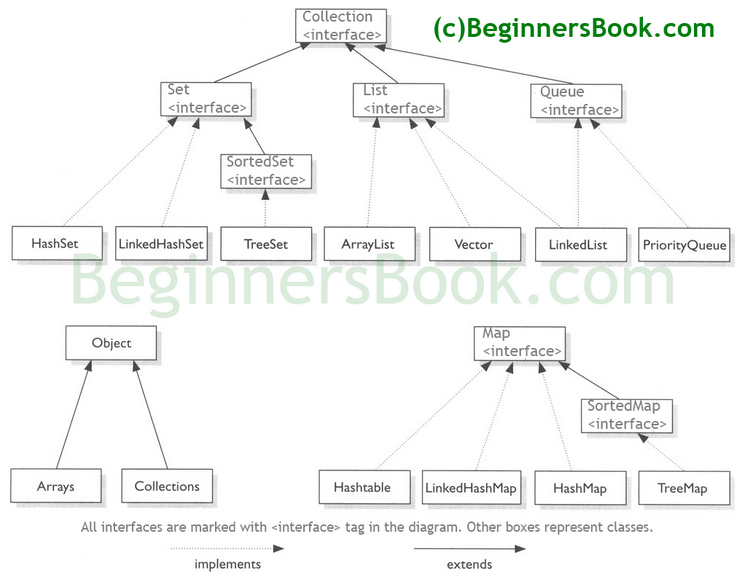
Two types of data structures are linear and non-linear.

* In Linear data structure data items are organized sequentially one after the other. Elements in data structure are traversed one after the other and only one element can be accessed one at a time.
* In non linear data items are not organized sequentially is called non linear. Data items could be connected to more than one element. All the elements in data structure cannot be traversed in single run.

**Collections in JAVA:**

* Collections framework is collection of interfaces and classes which helps in storing and processing data efficiently.

Collections framework Hierarchy:



**List Interface:**

* List is ordered collection of data.
* It may contain duplicate elements.
* Elements can be inserted or accessed by their position based on zero-based ind**ex.**

**Array List:**

* Array list class implements List interface.
* Array list is resizable array implementation of List interface.

**Array Vs ArrayList in java**

* Array is fixed size whereas array list is dynamic sized array in java
* Array is basic functionality provided by java , array List is part of collection of framework and it has set of methods to access the elements and modify them.
* Array can contain both primitive data types and objects of class depending on array definition.

However, array list can contain only objects of class it does not supports the primitive data types.

Note: When we do arrayList.add(1), it converts the primitive int data type to integer object.

* Since members of array list are always objects, content of array list is always reference to objects at different memory location. Therefore array list actual objects are never stored at contiguous memory location.

In case of primitive array elements are stored at contiguous locations, in case of objects similar to array list.

**Linked List:**

* It is linear data structure where the elements are not stored in contiguous memory and every element is separate object with data part and address part.
* Elements are linked using address part.
* Disadvantage of linked list is nodes cannot be accessed directly instead we need to start from the head and follow through link to reach the required the node.

Array list Versus Linked List:

* Array list is implemented with concept of dynamic array. However linked array list is implemented with concept of doubly linked list.
* Insertions are easy and fast in linked list, array list needs to update its index if element is inserted anywhere except at the end of array.
* Removal is faster is linked list.

Reason: Removal requires change in the pointer location in neighbor nodes of node which is going to be removed. In array list all the elements need to be shifted to fill out the space created by removing element.

* Linked list has more memory overhead than array list as each node needs to hold both data and address.
* Random access is not allowed in linked list.
* Array list search operation is fast compared to linked list.

Reason is as array list maintains the index based system for its elements which makes it faster for searching elements.

Similarities between array list and linked list are as follows,

1. Both ArrayList and LinkedList are implementation of List interface.
2. They both maintain the elements insertion order which means while displaying ArrayList and LinkedList elements the result set would be having the same order in which the elements got inserted into the List.
3. Both these classes are non-synchronized and can be made synchronized explicitly by using [Collections.synchronizedList](https://docs.oracle.com/javase/6/docs/api/java/util/Collections.html#synchronizedList(java.util.List)) method.
4. The iterator and list Iterator returned by these classes are fail-fast (if list is structurally modified at any time after the iterator is created, in any way except through the iterator’s own remove or add methods, the iterator will throw a [ConcurrentModificationException](https://docs.oracle.com/javase/6/docs/api/java/util/ConcurrentModificationException.html)).

**Set:**

* It is an interface which extends collection. Set is a collection that cannot contain the duplicate elements.
* It also adds a stronger contract on the behavior of equals and hash code operations, allowing set instances to be compared meaningfully.

**Hash Set:**

* It implements the set interface and it does not allow the duplicate elements.
* Underlying data structure is Hash map.
* It does not maintain the insertion order.
* It allows the Null values.
* Implements searlizable and clone able interfaces.
* Hash set uses the hash map internally to store the inserted elements, the values inserted is acts as key and java use a constant variable as its value of map.

**Tree Set:**

* It implements the sorted Set interface; it does not maintain insertion order.
* Objects in Tree set are sorted in ascending order.
* It throws classCastException if heterogeneous object is inserted.
* Tree set can be synchronized by synchronizing on some object which naturally encapsulates the set. Or it can be synchronized using Collections.synchronizedSet();
* Insertion of null into a TreeSet throws [NullPointerException](https://www.geeksforgeeks.org/null-pointer-exception-in-java/) because while insertion of null, it gets compared to the existing elements and null cannot be compared to any value.
* Secondly, if insertion of an object that cannot be compared with the elements existing in the set a ClassCastException is thrown.

Constructors of tree se are as below,

**TreeSet t = new TreeSet();**

**TreeSet t = new TreeSet(Comparator comp);**

**TreeSet t = new TreeSet(Collection col);**

**TreeSet t = new TreeSet(SortedSet s);**

**Hash Set versus Tree Set:**

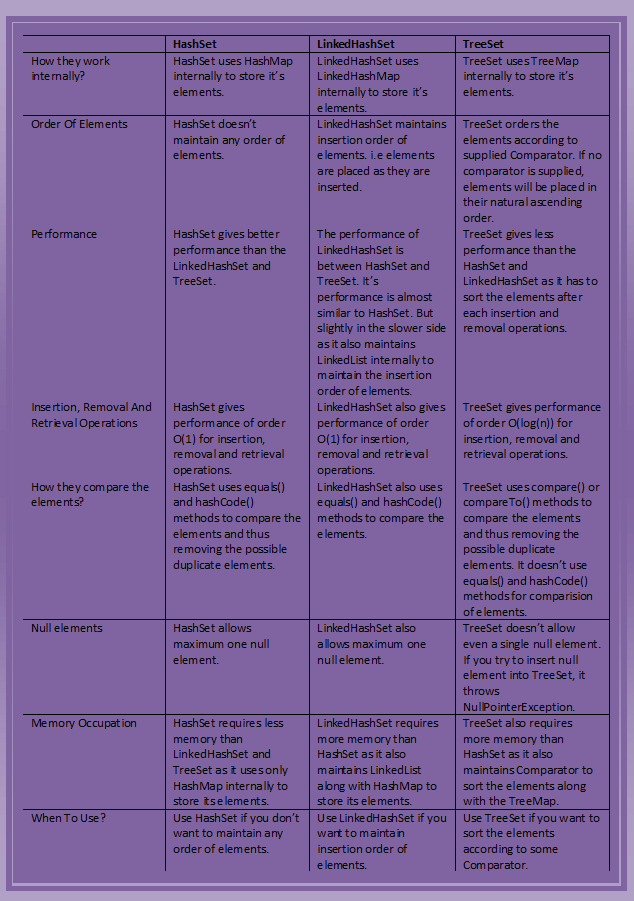
* First major difference between HashSet and TreeSet is performance. HashSet is faster than TreeSet and should be preferred choice if sorting of element is not required.
* Second difference between HashSet and TreeSet is that HashSet allows null object but TreeSet doesn't allow null Object and throw NullPointerException, Why, because TreeSet uses compareTo() method to compare keys and compareTo() will throw java.lang.NullPointerException.
* Another significant difference between HashSet and TreeSet is that , HashSet is backed by HashMap while TreeSet is backed by TreeMap in Java.
* One more difference between HashSet and TreeSet which is worth remembering is that HashSet uses equals() method to compare two object in Set and for detecting duplicates while TreeSet uses compareTo() method for same purpose. if equals() and compareTo() are not consistent, i.e. for two equal object equals should return true while compareTo() should return zero, than it will break contract of Set interface and will allow duplicates in Set implementations like TreeSet
* Now most important difference between HashSet and TreeSet is ordering. HashSet doesn't guaranteed any order while TreeSet maintains objects in Sorted order defined by either Comparable or Comparator method in Java.

The ordering maintained in each collections is as follows;

* HashSet - undefined.
* HashMap - undefined
* LinkedHashSet - insertion order
* LinkedHashMap - insertion order of keys (by default), or 'access order'
* ArrayList - insertion order.
* LinkedList - insertion order.
* TreeSet - ascending order, according to Comparable / Comparator.

**List Vs Set**

* List is an ordered collection it maintains the insertion order, which means upon displaying the list content it will display the elements in the same order in which they got inserted into the list.
* Set is an unordered collection, it doesn’t maintain any order. There are few implementations of Set which maintains the order such as LinkedHashSet (It maintains the elements in insertion order).
* List allows duplicates while Set doesn’t allow duplicate elements. All the elements of a Set should be unique if you try to insert the duplicate element in Set it would replace the existing value.
* List implementations: [ArrayList](https://beginnersbook.com/2013/12/java-arraylist/), [LinkedList](https://beginnersbook.com/2013/12/linkedlist-in-java-with-example/) etc.
* Set implementations: [HashSet](https://beginnersbook.com/2013/12/hashset-class-in-java-with-example/), [LinkedHashSet](https://beginnersbook.com/2013/12/linkedhashset-class-in-java-with-example/), [TreeSet](https://beginnersbook.com/2013/12/treeset-class-in-java-with-example/) etc.
* List allows any number of null values. Set can have only a single null value at most.
* [ListIterator](https://beginnersbook.com/2014/06/listiterator-in-java-with-examples/) can be used to traverse a List in both the directions(forward and backward) However it can not be used to traverse a Set. We can use [Iterator](https://beginnersbook.com/2014/06/java-iterator-with-examples/) (It works with List too) to traverse a Set.
* List interface has one legacy class called [Vector](https://beginnersbook.com/2013/12/vector-in-java/) whereas Set interface does not have any legacy class.



**MAP:**

* Map provides three collection views, set of keys (Set<String> keyset = map.keySet() ), set of key-value pairs(Set<Entry<String,String>> entrySet = map.entrySet();) and collection of values (Collection<String> values = map.values()).
* Map doesn’t guarantee the order of mappings, however it depends on the implementation. Example Tree map and linkedHashMap maintain the order where as hash map does not maintain.
* Map utilize hashcode and equals method on key for get and put operations. So mutable objects are not suitable for map keys. If the values of hash code equals change after put, you won’t get the correct value in get operation.
* Popular implementation of Map interface are, Treemap, Hashtable, Hashmap, ConcurrentHashMap and LinkedHashMap.
* AbstractMap class provides skeletal implementation of map interface.
* Map allows does not duplicate key whereas values can be duplicate.
* Some implementation allows null key and null value( Hashmap and LinkedHashMap) but some does not(Treeset).

**Hash Map:**

1. Java HashMap allows null key and null values.
2. HashMap is not an ordered collection. You can iterate over HashMap entries through keys set but they are not guaranteed to be in the order of their addition to the HashMap.
3. HashMap is almost similar to Hashtable except that it’s unsynchronized and allows null key and values.
4. HashMap uses it’s inner class Node<K,V> for storing map entries.
5. HashMap stores entries into multiple singly linked lists, called buckets or bins. Default number of bins is 16 and it’s always power of 2.
6. HashMap uses hashCode() and equals() methods on keys for get and put operations. So HashMap key object should provide good implementation of these methods. This is the reason [immutable](https://www.journaldev.com/129/how-to-create-immutable-class-in-java) classes are better suitable for keys, for example String and Interger.
7. Java HashMap is not thread safe, for multithreaded environment you should use ConcurrentHashMap class or get synchronized map using Collections.synchronizedMap() method.

**How HashMap works in java?**

HashMap works on the principle of hashing, we have put(key, value) and get(key) method for storing and retrieving Objects from HashMap. When we pass Key and Value object  to put() method on Java HashMap, HashMap implementation calls [hashCode method](http://javarevisited.blogspot.sg/2011/10/override-hashcode-in-java-example.html) on Key object and applies returned hashcode into its own hashing function to find a bucket location for storing Entry object, important point to mention is that HashMap in Java stores both key and value object as Map.Entry in a bucket.

**What will happen if two different objects have the same hashcode?**

Since hashcode is same, bucket location would be same and collision will occur in HashMap Since HashMap uses LinkedList to store object, this entry (object of Map.Entry comprise key and value )  will be stored in [LinkedList](http://javarevisited.blogspot.sg/2012/02/difference-between-linkedlist-vs.html).

**How will you retrieve Value object  if two Keys will have the same hashcode?**

we will call get() method and then HashMap uses Key Object's hashcode to find out bucket location. After finding bucket location, we will call keys.equals() method to identify a correct node in LinkedList and return associated value object for that key in Java HashMap.

**"What happens On HashMap in Java if the size of the HashMap  exceeds a given threshold defined by load factor ?"**

Java HashMap re-size itself by creating a new bucket array of size twice of the previous size of HashMap and then start putting every old element into that new bucket array. This process is called rehashing because it also applies the hash function to find new bucket location.

**"do you see any problem with resizing of HashMap  in Java"**

Yes there is potential [race condition](http://javarevisited.blogspot.sg/2012/02/what-is-race-condition-in.html) exists while resizing HashMap in Java, if two [thread](http://javarevisited.blogspot.sg/2011/02/how-to-implement-thread-in-java.html) at the same time found that now HashMap needs resizing and they both try to resizing. on the process of resizing of HashMap in Java, the element in the bucket which is stored in linked list get reversed in order during their migration to new bucket because Java HashMap  doesn't append the new element at tail instead it append new element at the head to avoid tail traversing. If race condition happens then you will end up with an infinite loop.

In case of collision, i.e, when there are different keys with same same hashcode, internally we use single linked list to store the elements. And we store every new element at the head of the linked list to avoid tail traversing and hence at the time of resizing the entire sequence of objects in linked list gets reversed, during which there are chances of infinite loops.

**Can we use ConcurrentHashMap in place of Hashtable?**

Since we know Hashtable is synchronized but ConcurrentHashMap provides better concurrency by only locking portion of map determined by concurrency level. ConcurrentHashMap is certainly introduced as Hashtable and can be used in place of it, but Hashtable provides stronger thread-safety than ConcurrentHashMap.

**How null key is handled in HashMap? Since equals() and hashCode() are used to store and retrieve values, how does it work in case of the null key?**  
The null key is handled specially in HashMap, there are two separate methods for that putForNullKey(V value) and getForNullKey(). Later is offloaded version of get() to look up null keys.  Null keys always map to index 0.  This null case is split out into separate methods for the sake of performance in the two most commonly used operations (get and put), but incorporated with conditionals in others. In short, equals() and hashcode() method are not used in case of null keys in HashMap.

**What is the difference between ConcurrentHashMap and Hashtable in Java?**

1) Hashtable is belongs to the Collection framework; ConcurrentHashMap belongs to the Executor framework.

2) Hashtable uses single lock for whole data. ConcurrentHashMap uses multiple locks on segment level (16 by default) instead of object level i.e. whole Map.

3) ConcurrentHashMap locking is applied only for updates. In case of retrievals, it allows full concurrency, retrievals reflect the results of the most recently completed update operations. So reads can happen very fast while writes are done with a lock.

4) ConcurrentHashMap doesn't throw a ConcurrentModificationException if one thread tries to modify it while another is iterating over it and does not allow null values.

5) ConcurrentHashMap returns Iterator, which fails-safe (i.e. iterator will make a copy of the internal data structure) on concurrent modification.

6) ConcurrentHashMap uses a database shards logic (Segment[] segments), i.e. divides the data into shards(segments) than puts locks on each shard (segment) instead of putting a single lock for whole data (Map).

7) ConcurrentHashMap is more efficient for threaded applications.

HashMap and Hashtable in Java:

* Hashtable is synchronized, whereas HashMap is not. This makes HashMap better for non-threaded applications, as unsynchronized Objects typically perform better than synchronized ones.
* Hashtable does not allow null keys or values. HashMap allows one null key and any number of null values.
* One of HashMap’s subclasses is LinkedHashMap, so in the event that you’d want predictable iteration order (which is insertion order by default), you could easily swap out the HashMap for a LinkedHashMap. This wouldn’t be as easy if you were using Hashtable.

**What’s the difference between ConcurrentHashMap and Collections.synchronizedMap(Map)?**

The main difference between these two is that ConcurrentHashMap will lock only portion of the data which are being updated while other portion of data can be accessed by other threads. However, Collections.synchronizedMap() will lock all the data while updating, other threads can only access the data when the lock is released. If there are many update operations and relative small amount of read operations, you should choose ConcurrentHashMap.  
   
Also one other difference is that ConcurrentHashMap will not preserve the order of elements in the Map passed in. It is similar to HashMap when storing data. There is no guarantee that the element order is preserved. While Collections.synchronizedMap() will preserve the elements order of the Map passed in. For example, if you pass a TreeMap to ConcurrentHashMap, the elements order in the ConcurrentHashMap may not be the same as the order in the TreeMap, but Collections.synchronizedMap() will preserve the order.  
   
Furthermore, ConcurrentHashMap can guarantee that there is no ConcurrentModificationException thrown while one thread is updating the map and another thread is traversing the iterator obtained from the map. However, Collections.synchronizedMap() is not guaranteed on this.

**What is the difference between a HashMap, ConcurrentHashMap and a Hashtable in Java?**

Hashtable:

Hashtable is synchronized, preventing two threads from accessing it at the same time. As of the Java 2 platform v1.2, this class was retrofitted to implement the Map interface, making it a member of the Java Collections Framework.

HashMap

If a thread-safe implementation is not needed, it is recommended to use HashMap in place of Hashtable.

HashMap is not synchronized. If multiple threads access a hash map concurrently, and at least one of the threads modifies the map structurally, it must be synchronized externally. (A structural modification is any operation that adds or deletes one or more mappings; merely changing the value associated with a key that an instance already contains is not a structural modification.)  
   
Synchronizing does not prevent ConcurrentModificationException, because they both say: The iterators returned by all of this class’s “collection view methods” are fail-fast: if the [Hashtable/map] is structurally modified at any time after the iterator is created, in any way except through the iterator’s own remove method, the iterator will throw a ConcurrentModificationException.  
   
Note that a single thread can cause ConcurrentModificationException by updating the Hashtable/HashMap directly while also iterating it. It doesn’t require multi-threading to violate this rule.

ConcurrentHashMap

If a thread-safe highly-concurrent implementation is desired, then it is recommended to use ConcurrentHashMap in place of Hashtable.  
   
In ConcurrentHashMap, Iterators and Enumerations return elements reflecting the state of the hash table at some point at or since the creation of the iterator/enumeration. They do not throw ConcurrentModificationException.  
   
However, iterators are designed to be used by only one thread at a time.  
   
So, HashMap is useful for single-threaded access. Hashtable is useful for multi-threaded access, as long as they don’t need to iterate the map. ConcurrentHashMap allows updates and iteration by multiple threads.

**Reference: <http://www.topjavatutorial.com/java-interview-questions/java-collections-interview-questions/>**