1.HashMap: HashMap offers 0(1) lookup and insertion. If you iterate through the keys, though, the ordering of the keys is essentially arbitrary. It is implemented by an array of linked lists.

Syntax:

public class HashMap extends AbstractMap

implements Map,Cloneable, Serializable

A HashMap contains values based on the key.

It contains only unique elements.

It may have one null key and multiple null values.

It maintains no order.

2. LinkedHashMap: LinkedHashMap offers 0(1) lookup and insertion. Keys are ordered by their insertion order. It is implemented by doubly-linked buckets.

Syntax:

public class LinkedHashMap extends HashMap

implements Map

A LinkedHashMap contains values based on the key.

It contains only unique elements.

It may have one null key and multiple null values.

It is same as HashMap instead maintains insertion order.

3.TreeMap: TreeMap offers O(log N) lookup and insertion. Keys are ordered, so if you need to iterate through the keys in sorted order, you can. This means that keys must implement the Comparable interface. TreeMap is implemented by a Red-Black Tree.

Syntax:

public class TreeMap extends AbstractMap implements

NavigableMap, Cloneable, Serializable

A TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.

It contains only unique elements.

It cannot have null key but can have multiple null values.

It is same as HashMap instead maintains ascending order(Sorted using the natural order of its key).

4. Hashtable: “Hashtable” is the generic name for hash-based maps.

Syntax:

public class Hashtable extends Dictionary implements

Map, Cloneable, Serializable

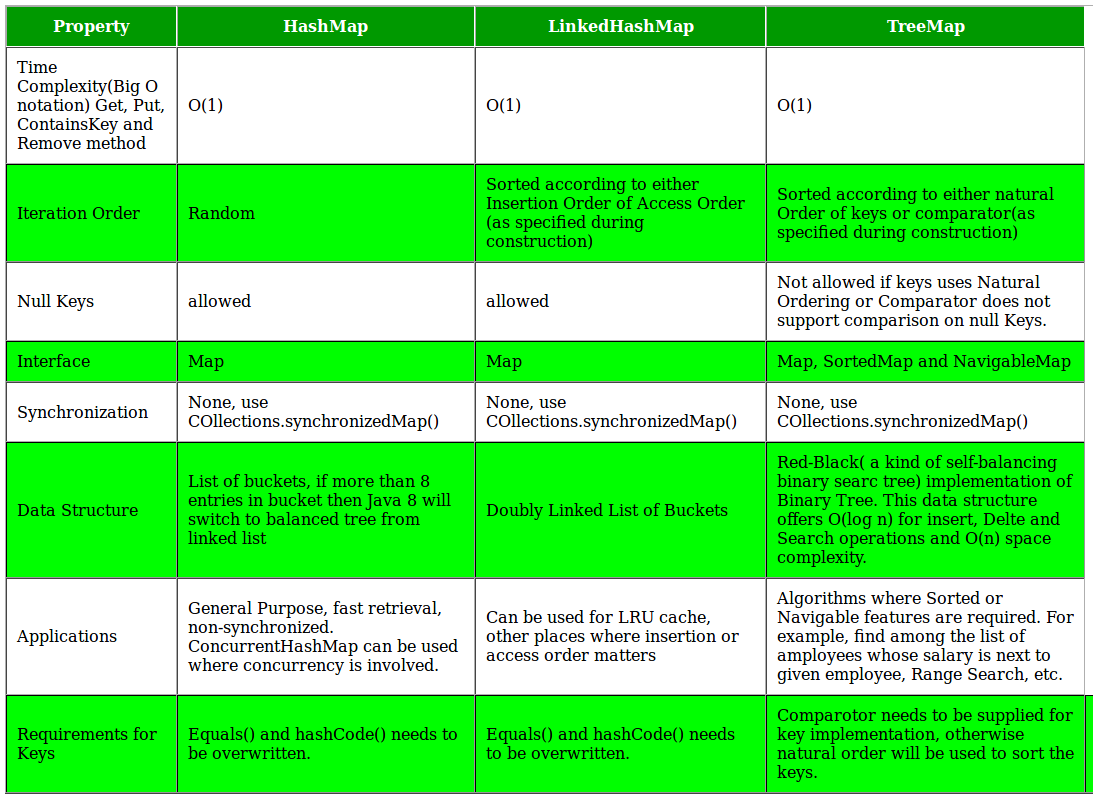
A Hashtable is an array of list. Each list is known as a bucket. The position of bucket is identified by calling the hashcode() method. A Hashtable contains values based on the key.

It contains only unique elements.

It may have not have any null key or value.

It is synchronized.

It is a legacy class.



**Key points of Concurrent HashMap:**

* The underlined data structure for Concurrent HashMap is [Hashtable](https://www.geeksforgeeks.org/hashtable-in-java/).
* Concurrent HashMap class is thread-safe i.e. multiple threads can operate on a single object without any complications.
* At a time any number of threads are applicable for a read operation without locking the Concurrent HashMap object which is not there in HashMap.
* In Concurrent HashMap, the Object is divided into a number of segments according to the concurrency level.
* The default concurrency-level of Concurrent HashMap is 16.
* In Concurrent HashMap, at a time any number of threads can perform retrieval operation but for updated in the object, the thread must lock the particular segment in which the thread wants to operate. This type of locking mechanism is known as **Segment locking or bucket locking**. Hence at a time, 16 update operations can be performed by threads.
* Inserting null objects is not possible in Concurrent HashMap as a key or value.

ArraList and Vector

Graphical user interface, application

Description automatically generated

Table

Description automatically generated

ArrayList and LinkedList

Text, table

Description automatically generated

| List | Set |
| --- | --- |
| 1. The List is an ordered sequence. | 1. The Set is an unordered sequence. |
| 2. List allows duplicate elements | 2. Set doesn’t allow duplicate elements. |
| 3. Elements by their position can be accessed. | 3. Position access to elements is not allowed. |
| 4. Multiple null elements can be stored. | 4. Null element can store only once. |
| 5. List implementations are ArrayList, LinkedList, Vector, Stack | 5. Set implementations are HashSet, LinkedHashSet. |

**Differences Between HashSet, LinkedHashSet**,**and TreeSet:**

| Features | HashSet | LinkedHashSet | TreeSet |
| --- | --- | --- | --- |
| Internal Working | HashSet internally uses HashMap for storing objects | LinkedHashSet uses LinkedHashMap internally to store objects | TreeSet uses TreeMap internally to store objects |
| When To Use | If you don’t want to maintain insertion order but want to store unique objects | If you want to maintain the insertion order of elements then you can use LinkedHashSet | If you want to sort the elements according to some Comparator then use TreeSet |
| Order | HashSet does not maintain insertion order | LinkedHashSet maintains the insertion order of objects | While TreeSet orders the elements according to supplied Comparator. By default, objects will be placed according to their natural ascending order. |
| Complexity of Operations | HashSet gives O(1) complexity for insertion, removing, and retrieving objects | LinkedHashSet gives insertion, removing, and retrieving operations performance in order O(1). | While TreeSet gives the performance of order O(log(n)) for insertion, removing, and retrieving operations. |
| Performance | The performance of HashSet is better when compared to LinkedHashSet and TreeSet. | The performance of LinkedHashSet is slower than TreeSet. It is almost similar to HashSet but slower because LinkedHashSet internally maintains LinkedList to maintain the insertion order of elements | TreeSet performance is better than LinkedHashSet except for insertion and removal operations because it has to sort the elements after each insertion and removal operation. |
| Compare | HashSet uses equals() and hashCode() methods to compare the objects | LinkedHashSet uses equals() and hashCode() methods to compare it’s objects | TreeSet uses compare() and compareTo() methods to compare the objects |
| Null Elements | HashSet allows only one null value. | LinkedHashSet allows only one null value. | TreeSet does not permit null value. If you insert null value into TreeSet, it will throw NullPointerException. |
| Syntax | HashSet obj=new HashSet(); | LinkedHashSet obj =new LinkedHashSet(); | TreeSet obj = new TreeSet(); |

**Difference Between Hashmap and Hashtable**

| S. No. | Hashmap | Hashtable |
| --- | --- | --- |
| 1. | No method is synchronized. | Every method is synchronized. |
| 2. | Multiple threads can operate simultaneously and hence hashmap’s object is not thread-safe. | At a time only one thread is allowed to operate the Hashtable’s object. Hence it is thread-safe. |
| 3. | Threads are not required to wait and hence relatively performance is high. | It increases the waiting time of the thread and hence performance is low. |
| 4. | Null is allowed for both key and value. | Null is not allowed for both key and value. Otherwise, we will get a null pointer exception. |
| 5. | It is introduced in the 1.2 version. | It is introduced in the 1.0 version. |
| 6. | It is non-legacy. | It is a legacy. |

There are two ways to create a Synchronized ArrayList.

1. Collections.synchronizedList() method.

2. Using CopyOnWriteArrayList.

**Runtime Error:**

Exception in thread "main" java.lang.UnsupportedOperationException