**HASHTABLE:**

Hashtable is an implementation of a key-value pair data structure in java. A ‘value’ can be stored and retrieved using a ‘key’ and it is [an identifier](https://javapapers.com/core-java/how-many-types-of-java-variables-are-there/) of the value stored. And the ‘key’ should be unique.

java.util.Hashtable extends Dictionary and implements Map. Objects with non-[null value](https://javapapers.com/core-java/nullpointerexception-null-bad-good-and-ugly/) can be used as a key or value. Key of the Hashtable must implement hashcode() and equals() methods.

Hashtable has two other constructors

Hashtable(int initialCapacity)&

Hashtable(int initialCapacity, float loadFactor)

----Initial capacity is number of buckets created at the time of Hashtable instantiation. Bucket is a logical space of [storage](https://javapapers.com/core-java/java-jvm-memory-types/)for Hashtable.

**Methods:**

1. **void clear() :** method clears the Hashtable so that it contains no keys.
2. **Object clone() :** used to create a shallow copy of this Hashtable.
3. **boolean contains(Object value) :** tests if some key maps into the specified value in this Hashtable.
4. **boolean containsValue(Object value) :**returns true if this hash table maps one or more keys to this value.
5. **Enumeration elements() :**Returns an enumeration of the values obtained in hash table.
6. **entrySet() :** used to get a set view of the entries contained in this hash table.
7. **boolean equals(Object o) :** used to compare specified object with this Map for equality.
8. **Object get(Object key) :** used to get the object that contains the value associated with key.
9. **int hashCode() :**returns the hash code value for this Map as per the definition in the Map interface.
10. **boolean isEmpty() :**used to test if this Hashtable maps no keys to values.
11. **Enumeration keys() :**used to get enumeration of the keys contained in the hash table.
12. **Object put(Object key, Object value) :**maps the specified key to the specified value in this Hashtable.
13. **KeySet() :**used to get a Set view of the keys contained in this hash table.
14. **void putAll(Map t) :** copies all of the mappings from the specified map to this Hashtable.
15. **protected void rehash() :** Increase the size of the hash table and rehashes all its keys.
16. **Object remove(Object key) :**Removes key and its value.
17. **int size() :**returns the number of entries in hash table.
18. **String toString() :**returns the string equivalent of a hash table.
19. **values() :**used to get a Collection view of the values contained in this Hashtable.

Find out the details about the data structure used by java.util.LinkedHashSet.

A LinkedHashSet is an ordered version of HashSet that maintains a doubly-linked List across all elements. LinkedHashSet iterate through the elements in the order in which they were inserted. When cycling through LinkedHashSet using an iterator, the elements will be returned in the order in which they were inserted. While iterating we can add or remove objects from the collection. LinkedHashSet provides collection of unique objects. LinkedHashSet is unsorted and non-indexed based collection class

 Find out the details about the data structure used by java.util.TreeSet.

java.util.TreeSet is implementation class of [SortedSet](https://www.geeksforgeeks.org/sortedset-java-examples/) Interface. TreeSet has following important properties.

1. TreeSet implements the [SortedSet](https://www.geeksforgeeks.org/sortedset-java-examples/) interface so duplicate values are not allowed.
2. TreeSet does not preserve the insertion order of elements but elements are sorted by keys.
3. TreeSet does not allow to insert Heterogeneous objects. It will throw classCastException at Runtime if trying to add heterogeneous objects.
4. TreeSet is basically implementation of a self-balancing binary search tree like [Red-Black Tree](https://www.geeksforgeeks.org/red-black-tree-set-1-introduction-2/). Therefore, operations like add, remove and search take O(Log n) time. And operations like printing n elements in sorted order takes O(n) time.