**HashSet & LinkedHashSet**

Both implement Set interface. However in HashSet implemented using a hash table. The elements are stored in buckets based on their hash codes, allowing for fast insertion, deletion, and lookup operations with an average time complexity of O(1) for each operation. This means we can never get a certain order in which the elements are stored as they are inserted in.

LinkedHashSet is implemented as a combination of a hash table and a doubly-linked list. In addition to the hash table for fast lookups, it maintains the order of elements based on their insertion. This means that the elements are stored in the order they were added, enabling the iteration of elements in the order of insertion.

In HashSet the iteration order of elements in a HashSet is not predictable. It depends on the internal hash code-based organization and may change when the set is resized. But in LinkedHashSet the iteration order of elements in a LinkedHashSet is the order of insertion. It guarantees that elements will be iterated in the same order they were added.

HashSet set generally has better performance for adding, removing, and checking for element existence due to its O(1) average time complexity for the operations. LinkedHashSet has slightly slower performance for adding and removing elements compared to HashSet because it needs to maintain the linked list to preserve the insertion order. However, the difference is usually negligible for small sets.

Memory usage is low as compare to LinkedHashSet as it doesn’t new an LinkedList of the elements. We should only use LinkedHashSet when we want to preserve the order of insertion of the elements in the set.

TreeSet Comparison Methods:

N0 TreeSet doesn’t use equals and hashcode() methods for comparison. Instead it uses the natural ordering if the elements implement the Comparable interface or a custom comparator if provided during the TreeSet creation to determine the order of elements in the set. The compareTo() method of the Comparable interface or the compare() method of the Comparator interface is used.

Time Complexities:

The time complexity of various operations for TreeSet, HashSet, and LinkedHashSet in Java are as follows:

TreeSet: - Add: O(log n)

-> Remove: O(log n)

-> Contains: O(log n)

-> Accessing elements using iterator: O(n) for full iteration (traversing all elements)

-> Searching for minimum and maximum elements: O(1)

The operations like add, remove, and contains in TreeSet have a time complexity of O(log n) because it uses a self-balancing binary search tree (e.g., Red-Black Tree or AVL Tree) internally. The time complexity of iterator-based operations is O(n) because traversing all elements requires visiting each node in the tree.

HashSet:

-> Add: O(1) average case, O(n) worst case

-> Remove: O(1) average case, O(n) worst case

-> Contains: O(1) average case, O(n) worst case

-> Accessing elements using iterator: O(n) for full iteration (traversing all elements)

The time complexity for HashSet is O(1) for average-case add, remove, and contains operations, assuming a good hash function and minimal collisions. However, in the worst case (rare), when there are many hash collisions, these operations can degrade to O(n), where n is the number of elements in the set.

LinkedHashSet:

-> Add: O(1) average case, O(n) worst case (due to potential resizing)

-> Remove: O(1) average case, O(n) worst case (due to potential resizing)

-> Contains: O(1) average case, O(n) worst case (due to potential resizing)

-> Accessing elements using iterator: O(n) for full iteration (traversing all elements)

The LinkedHashSet is similar to HashSet in terms of time complexity for add, remove, and contains operations, with O(1) average-case complexity but O(n) worst-case complexity in case of potential resizing. The main difference between LinkedHashSet and HashSet is that LinkedHashSet maintains the order of insertion, making it slightly slower due to maintaining the linked list.

HashMap Working

It internally implements a HashMap. An element can be inserted into the HashSet using the add function. This internally calls the put() function since a HashMap would have been internally created. Hence, Set takes in unique values with the help of HashMap. HashMap contains unique key and value pairs, wherein the key and value pairs are inserted using the put() function. Upon calling the put() function, a previous value associated with the key or null is returned depending on whether a mapping is present for the key or not. LinkedHashSet extends to the HashSet class, meaning LinkedHashSet calls the constructors of HashSet class using the super() function.