Java Collection Framework:

Collection is set of element

* **List**: An ordered collection that can contain duplicate elements.

**Ex**:ArrayList and LinkedList.

* **Set**: A collection that does not allow duplicates.

**Ex**: HashSet, TreeSet, and LinkedHashSet.

* **Map**: A collection of key-value pairs. Keys are unique, and values can be duplicated.

Ex:HashMap, TreeMap, and LinkedHashMap.

* **Queue**: A collection designed for holding elements prior to processing. PriorityQueue and LinkedList are common implementations.



What makes HashSet an unique collection?

equals() and hashcode() method

What makes TreeSet an unique collection

Compareto method of comparable interface

Iterator:

An **Iterator** in Java is an interface that provides a way to traverse through a collection (like lists, sets, etc.) without exposing the underlying structure of the collection.

**Key Methods of the Iterator Interface**

1. **boolean hasNext()**: Returns true if there are more elements in the collection to iterate over.
2. **E next()**: Returns the next element in the iteration. If no more elements are present, it throws a NoSuchElementException.
3. **void remove()**: Removes the last element returned by the iterator. This method can be called only once per call to next().

**What differences exist between Iterator and ListIterator ?**

The differences of these elements are listed below:

• An Iterator can be used to traverse the Set and List collections, while the ListIterator can be used to iterate only over Lists.

• The Iterator can traverse a collection only in forward direction, while the ListIterator can traverse a List in both directions.

• The ListIterator implements the Iterator interface and contains extra functionality, such as adding an element, replacing an

element, getting the index position for previous and next elements, etc

**How Does a HashMap Work?**

1. **Storing Keys**:
   * When you add a key-value pair to a HashMap, it uses the hashCode() method of the key to figure out where to store that pair. The hashCode() gives a number (the hash code) that indicates the location (or "bucket") in the HashMap where the data should go.
2. **Handling Collisions**:
   * Sometimes, two different keys can produce the same hash code. This is called a "collision."
   * To handle this, the HashMap uses the **equals()** method to check if the keys are actually the same. If two keys have the same hash code, equals() helps determine if they refer to the same key or if they are different keys that just happened to land in the same spot.
   * -----------------
3. **What is the importance of hashCode() and equals() methods ?**

In Java, a HashMap uses the hashCode and equals methods to determine the index of the key-value pair and to detect duplicates.

More specifically, the hashCode method is used in order to determine where the specified key will be stored. Since different keys may produce the same hash value, the equals method is used, in order to determine whether the specified key actually exists in the collection or not. Therefore, the implementation of both methods is crucial to the accuracy and efficiency of the HashMap

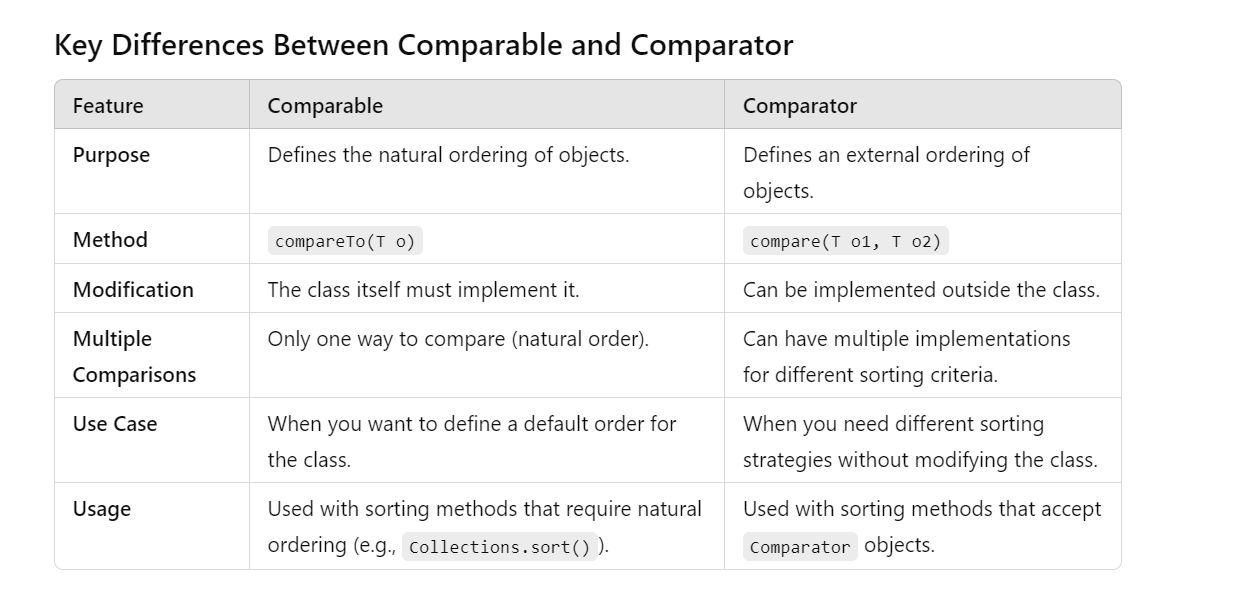
**What is Comparable and Comparator interface ? List their differences.**

* Java provides the Comparable interface, which contains only one method, called **compareTo.**

This method compares two objects,in order to impose an order between them. Specifically, it returns a negative integer, zero, or a positive integer to indicate that the input object is less than, equal or greater than the existing object.

* Java provides The Comparator interface, which contains two methods, called **compare and equals**. The first method compares its two input arguments and imposes an order between them.

It returns a negative integer, zero, or a positive integer to indicate that the first argument is less than, equal to, or greater than the second. The second method requires an object as a parameter and aims to decide whether the input object is equal to thecomparator. The method returns true, only if the specified object is also a comparator and it imposes the same ordering as the comparator.



|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| 1) ArrayList internally uses a **dynamic array** to store the elements. | LinkedList internally uses a **doubly linked list** to store the elements. |
| 2)(Insertion,deletion is slower)  Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the other elements are shifted in memory. | Insertion,deletion is faster  Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory. |
| 3) An ArrayList class can **act as a list** only because it implements List only. | LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces. |
| 4) ArrayList is **better for storing and accessing** data. | LinkedList is **better for manipulating** data. |
| 5) The memory location for the elements of an ArrayList is contiguous. | The location for the elements of a linked list is not contagious. |
| 6) Generally, when an ArrayList is initialized, a default capacity of 10 is assigned to the ArrayList. | There is no case of default capacity in a LinkedList. In LinkedList, an empty list is created when a LinkedList is initialized. |
| 7) To be precise, an ArrayList is a resizable array. | LinkedList implements the doubly linked list of the list interface. |

Hashset : it does not allow dublicate elements,unique elements,allows one null element.

What makes hashset unique? -> eqals and hashcode method

Treeset:it is in sorted order,no doublicates will be allowed , Does not allow null elements (throws NullPointerException)

What makes treeset unqique?-> compareto method of comparable interface

**Key Differences Between HashSet and TreeSet**

| **Feature** | **HashSet** | **TreeSet** |
| --- | --- | --- |
| **Implementation** | Uses a hash table | Uses a Red-Black tree (self-balancing binary search tree) |
| **Ordering** | Unordered; no guarantee of order | Ordered; elements are sorted based on their natural ordering or a specified comparator |
| **Time Complexity** | - **Add**: O(1) average case - **Remove**: O(1) average case - **Contains**: O(1) average case | - **Add**: O(log n) - **Remove**: O(log n) - **Contains**: O(log n) |
| **Null Elements** | Allows one null element | Does not allow null elements (throws NullPointerException) |
| **Duplicates** | Does not allow duplicates | Does not allow duplicates |
| **Use Case** | Ideal for fast lookups and when order is not important | Ideal for sorted data and when maintaining order is necessary |

**Concurrent Collections**

Java also provides specialized collections for multi-threaded environments, which allow thread-safe operations without external synchronization. Some important classes are:

* **ConcurrentHashMap**: A thread-safe map implementation that allows concurrent reads and updates.
* **CopyOnWriteArrayList**: A thread-safe list that creates a copy of the list on every write operation.

Hashmap and hashtable diff

Treeset