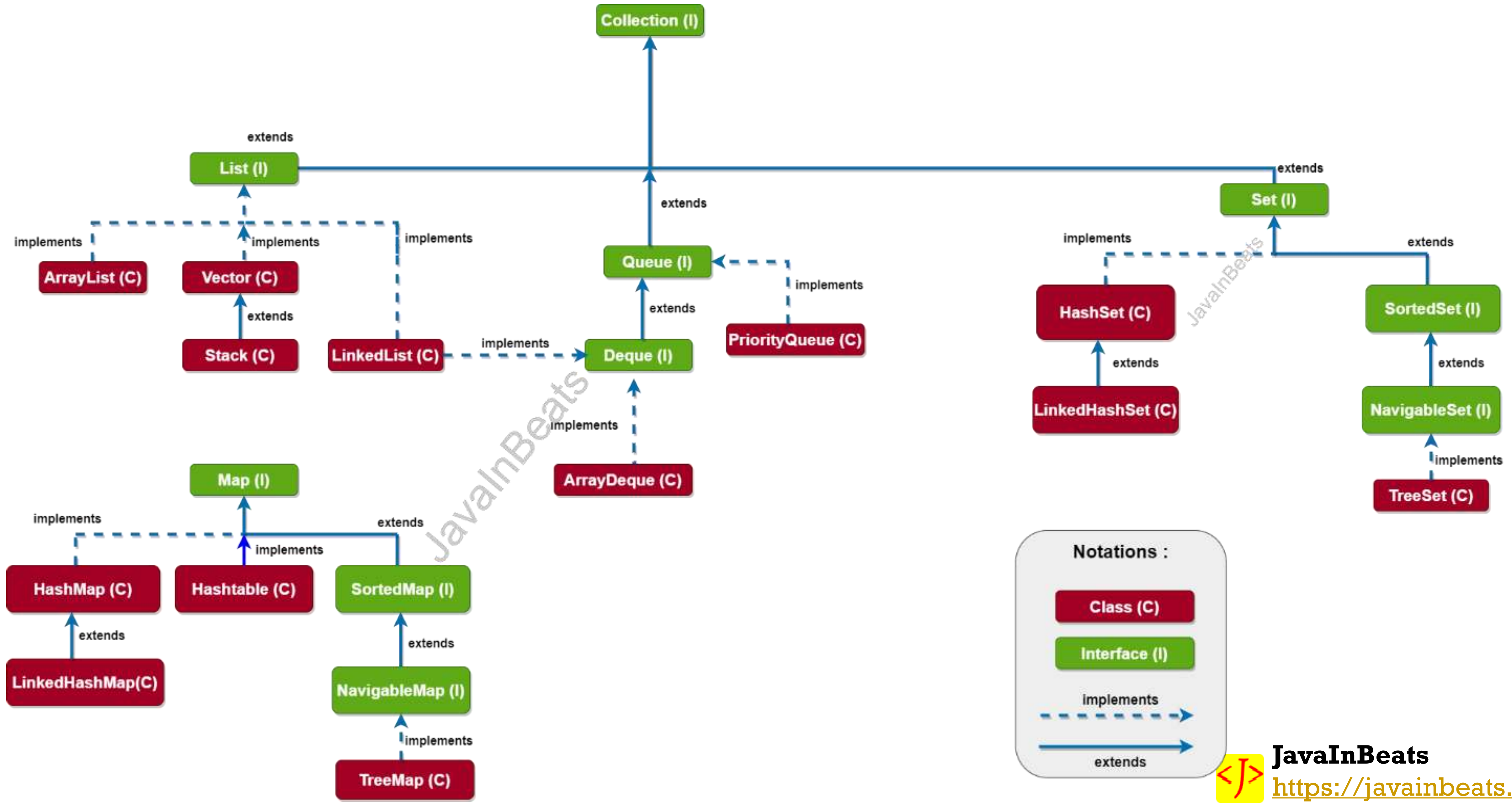
COLLECTION FRAMEWORK

WHAT IS COLLECTION?

1. Collection is use to store an elements (Object) of same or different type.
2. Using this you can create a group of object/element.
3. Collection implemented classes are based on the Data Structure (DS).
4. You can group the elements based on multiple criteria using collection.
5. In collection framework there are multiple classes and interfaces using which you can achieve a functionalities/method in the group of values.
6. All the classes and interfaces(APIs) are present inside **java.util package**.
7. There are 3 types of collection
   a. **List:** Allows duplicate elements/Object.
   b. **Queue:** Allows elements based on First On First Out (FIFO)
   c. **Set:** Allows unique elements/Object.

COLLECTION HIERARCHY

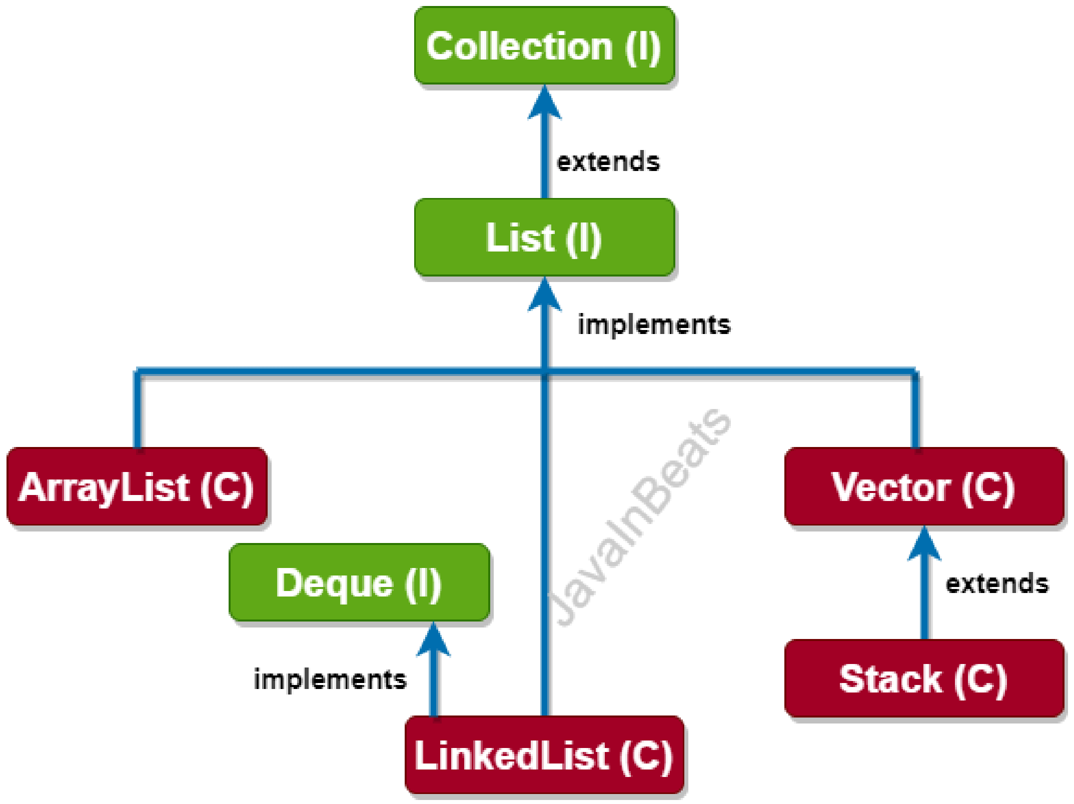


COMMON COLLECTION METHODS

Method	Description
add()	Add method is use to insert the element into collection.
remove()	Remove the single matching value from the collection.
contains()	This method is use to check where the element present inside collection or not.
size()	This method is to get total number of elements present into Collection.
clear()	This method is use to remove all the elements from the collection.
addAll()	This method is use to insert multiple element at a same time in a collection.
removeAll()	Remove multiple elements at a same time from the collection.
containsAll()	This method is use to check where the given elements present inside collection or not.
iterator()	Method to iterate the elements form the collection one by one,
isEmpty()	This method is use to check whether collection is empty or not.
retainAll()	retains only those elements of an ArrayList that are present in the Collection that is passed to the method.

LIST INTERFACE

1. It is an one of the type of collection.
2. List interface is a child interface of Collection.
3. List is an indexed based collection.
4. List allows duplicate elements.
5. List can store values different type of object having dynamic in size.
6. Items can be retrieved and inserted at specific positions in the list based on an index much like an array.
7. There are implemented classes of the List
    a. **ArrayList**
    b. **Vector**
    c. **LinkedList**



ARRAYLIST CLASS

1. ArrayList is backed by array, that is internal data structure is dynamic Array.
2. All the elements of ArrayList stores at a specific index.
3. Elements store in the ArrayList are in insertion order.
4. Can store element of any data type, it is also known as heterogeneous collection of elements.
5. Can store duplicate elements.
6. ArrayList is non-synchronized.
7. Random Access is allowed.
8. Default initial capacity of ArrayList is "10".
9. You can create ArrayList with customized capacity, by using following way.
   ArrayList list = new ArrayList(int initialcapacity);
10. Every Random insertion and deletion operation on ArrayList element causes a shifting index of other elements and hence these operations are slower in ArrayList.

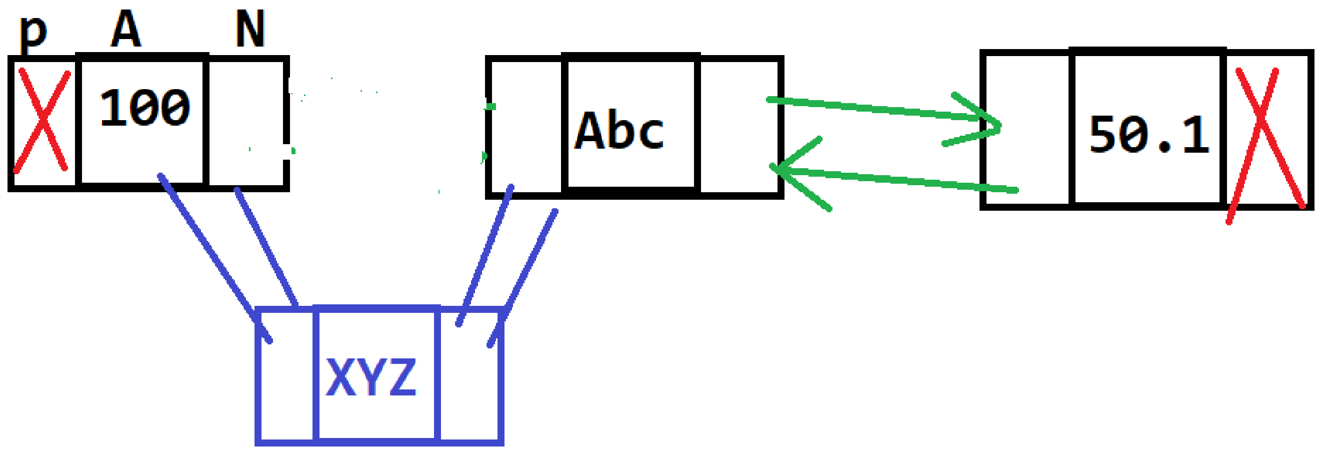
```
ArrayList list = new ArrayList();
list.add(123);
list.add("Abc");
list.add(12.34);
list.add('A');
list.add(123);
list.add(true);
list.add("Abc");
```

index	0	1	2	3	4	5	6
Value	123	Abc	12.34	A	123	true	ABC

JavaInBeats
https://javainbeats.com/

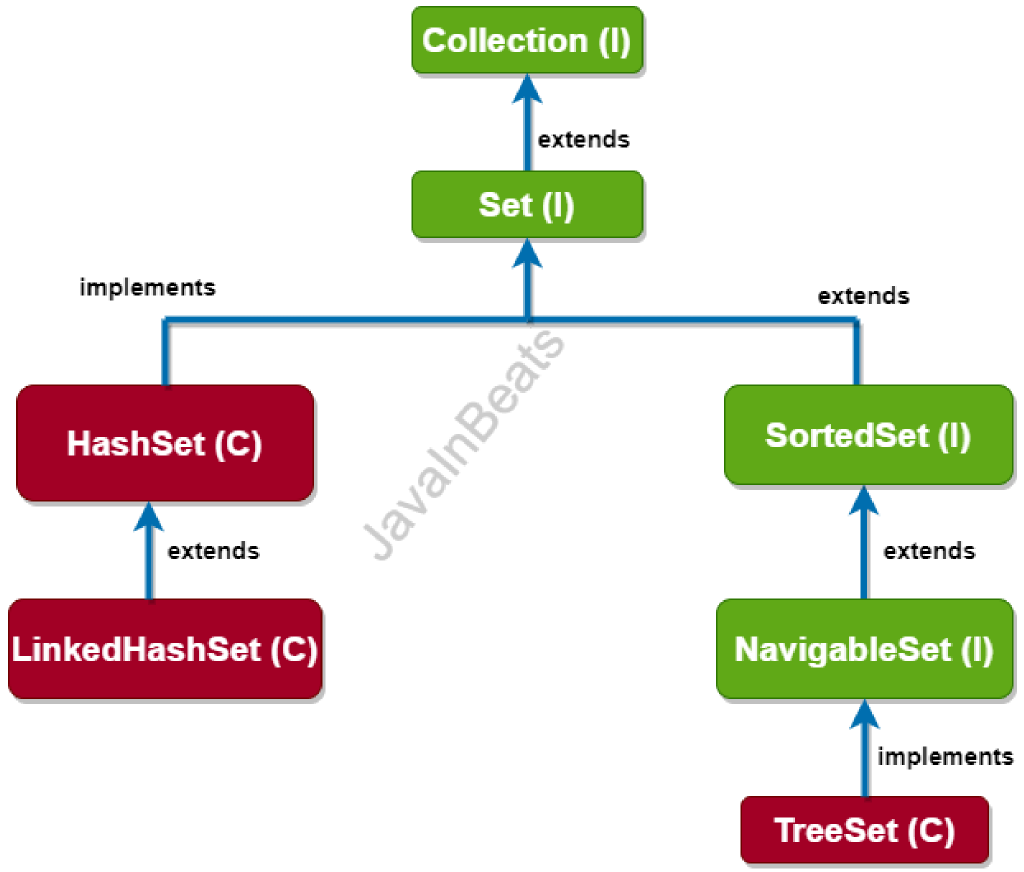
LINKEDLIST CLASS

1. Linked List is a collection heterogeneous element.
2. Can store duplicate elements.
3. Linked List based on doubly linked list data structure.
4. Linked list maintain the insertion order.
5. Each element in the linked list maintains the details of previous and next element.
6. Linked list provides faster insertion or deletion operation.
7. If your frequent operation is insertion (or) deletion in the middle then Linked List is the best choice.
8. If your frequent operation is retrieval operation then Linked List is worst choice. Because it provides slower iteration.
9. Linked List implements the property from List, Deque interface.



SET INTERFACE

1. Set is a type of collection.
2. Set interface is a child interface of Collection.
3. Set is non-index based.
4. Set is a collection on unique values.
5. Set allows different type of values and it can be dynamic is size.
6. Set Implemented Classes
   a. **HashSet**
   b. **LinkedHashSet**
   c. **TreeSet**



JavaInBeats
https://javainbeats.com/

HASHSET CLASS

1. Hash Set can store heterogeneous elements.
2. It is in dynamic in size.
3. In HashSet elements are store in unordered and unsorted way.
4. HashSet store only unique elements
5. HashSet is based on Hashtable data structure.
6. Can store null values (only one null value at a time).
7. HashSet internally use hashing mechanism to store elements.
8. Searching operation is faster in HashSet.
9. Default capacity of HashSet is 16 and 0.75 load factor.
10. HashSet is non-synchronized.
11. In HashSet RandomAccess is not.

LINKEDHASHSET CLASS

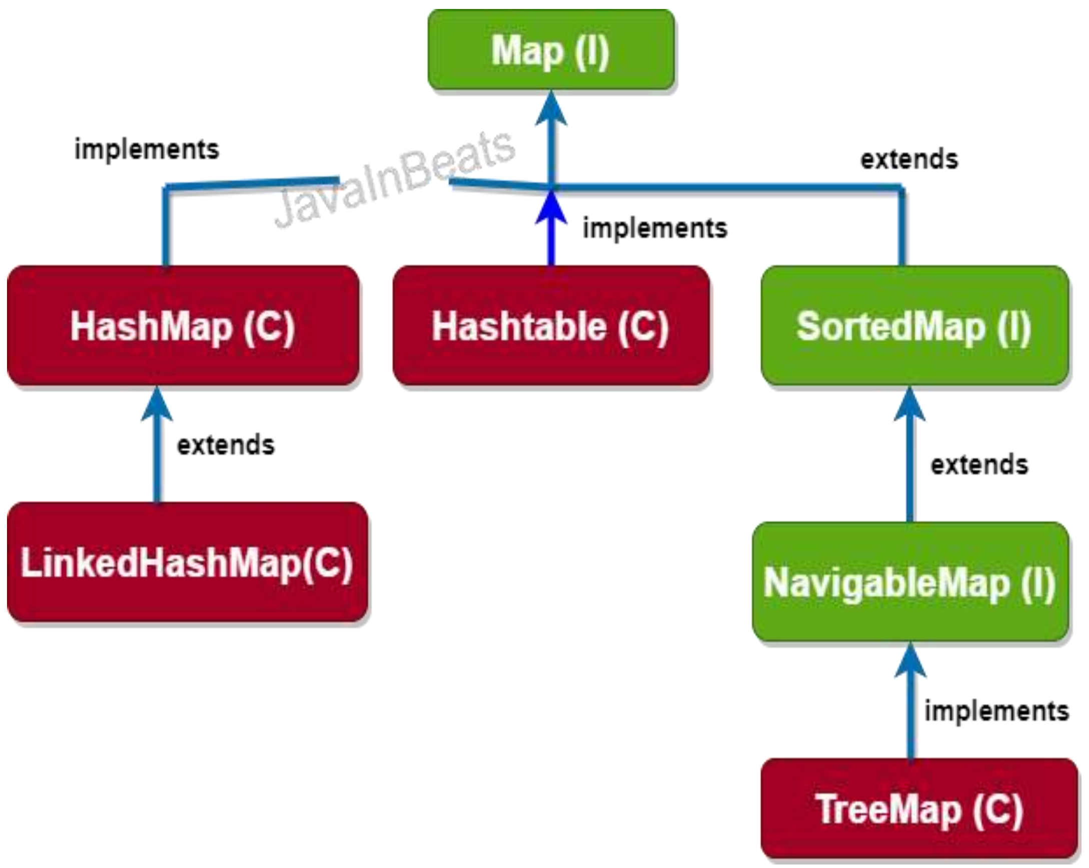
1. LinkedHashSet can store heterogeneous elements.
2. It is in dynamic in size.
3. LinkedHashSet maintain insertion order.
4. LinkedHashSet store only unique elements
5. LinkedHashSet is based on LinkedList and Hashtable data structure.
6. Can store null values (only one null value at a time).
7. HashSet internally use hashing mechanism to store elements.
8. Searching and updation operation is faster in HashSet.
9. Default capacity of HashSet is 16 and 0.75 load factor.
10. LinkedHashSet is non-synchronized.
11. In LinkedHashSet RandomAccess is not.

TREESET CLASS

1. TreeSet can store homogeneous elements only.
2. It is in dynamic in size.
3. Is not indexed base.
4. TreeSet store only unique elements
5. TreeSet implements the property from Set, SortedSet, NavigableSet interface.
6. Based on Balanced Tree algorithm.
7. Stores elements in sorted way.
8. The Class Object which added into TreeSet, that class must implements Comparable interface or must pass Comparator implemented class.
9. The Comparator and Comparable are the two interfaces use to provide sorting logic for TreeSet.

MAP INTERFACE

1. Map is not considered as a part of collection because it not implements the property from collection interface.
2. Map Store entry inside it. Once entry is also known as one key and value pair.
3. Each key must be unique and values can be duplicate.
4. No size limit to store key and value pair.
5. Key and value can be of any data type.
6. Map is an interface inside java.util.
7. Implemented classes of map are as follows
   a. Hashtable
   b. HashMap
   c. LinkedHashMap
   d. TreeMap



COMMON MAP METHODS

Method	Description
put(Object k, Object V)	Is use to add single key and value pair(entry) inside map.
remove(Object k)	Is use to get the value from the map.
containsKey(Object k)	Check whether key is present inside map or not.
containsValue(Object v)	Check whether value is present inside map or not.
size()	To get total number of entries present inside map..
isEmpty()	Is use to check whether map is empty or not.
putAll(Map)	Is use to add multiple entries inside map.
keySet()	Is use to get all the keys from the map.
values()	Is use to get all values from the map.
clear()	Is use to remove all the entries from the map.
get(Object)	Is use to get the value of specific key

HASHMAP CLASS

1. HashMap implements the property of Map interface.
2. It is use to store key and value pair.
3. Dynamic in size.
4. Can store keys and values of any data types.
5. Each key in map must be unique and values can be duplicate.
6. HashMap is based on Hashing algorithm.
7. HashMap is unorder and unsorted.
8. Can store one null key and multiple null values.
9. Default capacity is 16 and load factor is 0.75

HASHTABLE CLASS

1. Hashtable implements the property of Map interface.
2. It is use to store key and value pair
3. Dynamic in size.
4. Can store keys and values of any data types.
5. Each key in map must be unique and values can be duplicate.
6. Hashtable is based on Hashing algorithm.
7. Hashtable is unorder and unsorted.
8. Hashtable is available from JDK 1.0 version.
9. Hashtable is also known as legacy class.
10. Hashtable Methods are synchronized.
11. Hashtable object is thread-safe.
12. Hashtable is slower than HashMap.
13. Null key or null values are not allowed.

LINKEDHASHMAP CLASS

1. LinkedHashMap implements the property of Map interface and also extends HashMap class
2. It is use to store key and value pair.
3. Dynamic in size.
4. Can store keys and values of any data types.
5. Each key in map must be unique and values can be duplicate.
6. LinkedHashMap is based on Hashing and doubly linked list algorithm.
7. LinkedHashMap maintains insertion order.
8. Can store one null key and multiple null values.
9. Default capacity is 16 and load factor is 0.75

TREEMAP CLASS

1. TreeMap implements the properties from Map , NavigableMap and SortedMap interface.
2. It is use to store key and value pair
3. Dynamic in size.
4. Keys must be of same data type and values can be of any data types.
5. Each key in map must be unique and values can be duplicate.
6. TreeMap is based on Balance Tree algorithm.
7. TreeMap sorts the entries by key.
8. Cannot store null key, but can have null values.

ITERATING COLLECTION

1.Iterating Collection using Iterator Interface

- You can get values form one by one from collection by using Iterator.
- Can Iterate collection by **forward Direction** only.
- Common Methods from Iterator.

Method	Description
hasNext()	This method is use to check whether next element is present inside collection or not. Returns true if next element is present else return false
next()	Is use to return element present inside next location.
remove()	Is use to remove current location element. By remove, it will delete the element from the list.

ITERATING COLLECTION

1.Iterating Collection using ListIterator Interface

- You can get values form one by one from collection by using ListIterator.
- Can Iterate collection by **forward as well as backward/reverse Direction**.
- ListIterator extends the property from Iterator Interface.
- Common Methods from ListIterator.

Method	Description
hasNext()	This method is use to check whether next element is present inside collection or not. Returns true if next element is present else return false
next()	Is use to return element present inside next location.
remove()	Is use to remove current location element. By remove, it will delete the element from the list.
hasPrevious()	This method is use to check whether previous element is present inside collection or not. Returns true if previous element is present else return false.
previous()	Is use to return element present inside previous location.
add(E)	Is use to replace new element with current element.
set(E)	Is use to replace new element with current element.