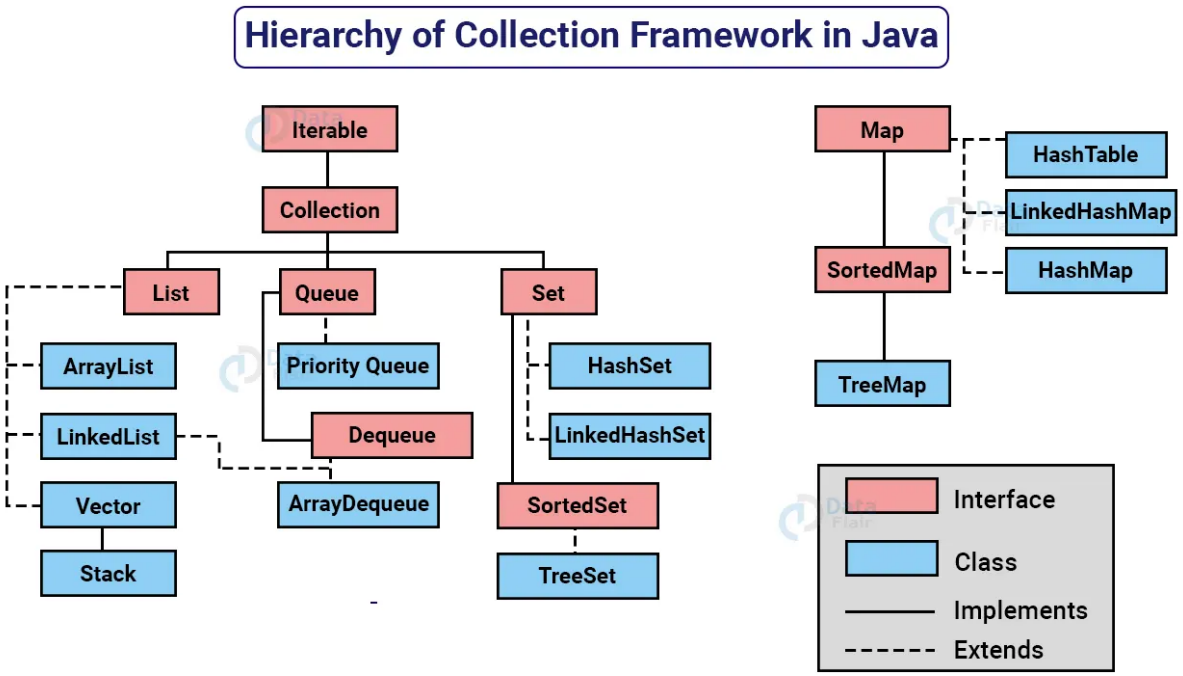
COLLECTIONS

Collection is like a container, an object that groups multiple elements into to a single unit. Collections are used to store, retrieve, manipulate, and communicate aggregate data. They represent data items that form natural group.

Ex: Telephone directory, Mail folder, Poker cards.

Collection hierarchy:



1. What is the collections framework?

Is a unified architecture for representing and manipulating collections. Collection framework contains interfaces, implementations, and algorithms.

- *Interfaces:* Allow collections to be manipulated independently of the details of their representations.
- *Implementation:* These are the concreate implementations of the collection interface, they are reusable data structure.
- *Algorithms:* These are methods to perform operations such sorting and searching.

2. What are the benefits of collections framework?

1) ***Reduces programming efforts:*** provides useful data structures and algorithms to manipulate collections.

2) ***Increases programming speed and quality:*** collections framework provides high-performance, high-quality data structures and algorithms to manipulate.

3) ***Allows interoperability among unrelated API's:*** collections frameworks provide interoperability among unrelated APIs to perform operations.

4) ***Reduces effort to learn and use new API's:*** earlier there are lot of APIs are present to work with collections but know by using efficient APIs this problem resolved.

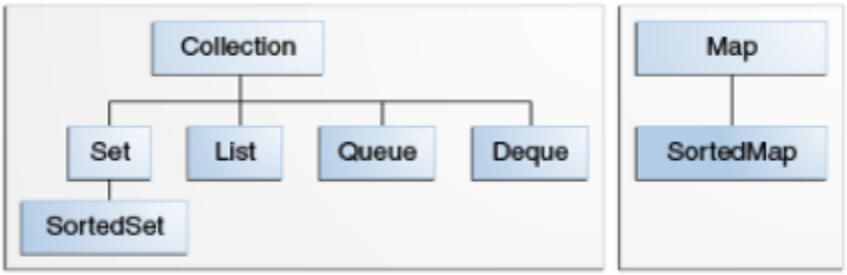
5) ***Reduces effort to design new API's:*** no need to design new APIs to work with collections instead of using standard collections interfaces.

6) ***Foster software reuse:***

3. Explain Collections class?

java.util.Collections is a class consisting of static methods (algorithms) to perform on the collections. This class contains methods for algorithms, like binary sorting, search, shuffling etc.

4. Explain Collection interface?



The core collection interfaces.

Collection interface is the root hierarchy of the java collection framework. It encapsulates different types of interfaces that are independent of their representation. Collection interface provides methods to perform operations on the collections.

5. Explain Iterator interface?

Iterator is an interface that contains two abstract methods and one non-return (void) method like hasNext(), next() and remove() respectively. This interface allows collections to iterate over the collection. There are three ways to iterate a collection they are Aggregate operation, for each Construct, and iterator.

Iterator provides the iterator method to iterate the collection and provides the remove() method to remove the element that satisfies the condition during iteration.

6. Explain Set interface?

A Set is a collection interface that restricts the duplicate elements stored in the collection. It does not maintain insertion order of the elements. It contains methods that are inherited from the Collection interface and uses those methods to restricts duplicate elements by using hashCode() and equals() methods.

Set interface has three implementations:

- HashSet: It stores elements in hash table, it makes no guarantees concerning order of the iteration.
- TreeSet: It stores elements in red-black tree, orders based on the values of the elements.
- LinkedHashSet: Which is implemented as a hash table with linked list running through it. It stores elements based on the insertion order.

7. What is ArrayList?

ArrayList is a data structure used to elements that have dynamic behavior i.e., it can grow its size by adding elements and can shrink size by removing elements.

8. What are the methods of Collection?

Modifier and Type	Method and Description
boolean	**add**(E e) Ensures that this collection contains the specified element (optional operation).
boolean	**addAll**(Collection<? extends E> c) Adds all of the elements in the specified collection to this collection (optional operation).
void	**clear**() Removes all of the elements from this collection (optional operation).
boolean	**contains**(Object o) Returns true if this collection contains the specified element.
boolean	**containsAll**(Collection<?> c) Returns true if this collection contains all of the elements in the specified collection.
boolean	**equals**(Object o) Compares the specified object with this collection for equality.
int	**hashCode**() Returns the hash code value for this collection.
boolean	**isEmpty**() Returns true if this collection contains no elements.
Iterator<E>	**iterator**() Returns an iterator over the elements in this collection.
default Stream<E>	**parallelStream**() Returns a possibly parallel Stream with this collection as its source.
boolean	**remove**(Object o) Removes a single instance of the specified element from this collection, if it is present (optional operation).
boolean	**removeAll**(Collection<?> c) Removes all of this collection's elements that are also contained in the specified collection (optional operation).
default boolean	**removeIf**(Predicate<? super E> filter) Removes all of the elements of this collection that satisfy the given predicate.
boolean	**retainAll**(Collection<?> c) Retains only the elements in this collection that are contained in the specified collection (optional operation).
int	**size**() Returns the number of elements in this collection.
default Spliterator<E>	**spliterator**() Creates a Spliterator over the elements in this collection.
default Stream<E>	**stream**() Returns a sequential Stream with this collection as its source.
Object[]	**toArray**() Returns an array containing all of the elements in this collection.
<T> T[]	**toArray**(T[] a) Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of the specified array.

9. Explain List interface?

List is the data structure of the collection framework. List stores multiple items based on insertion order; it extends methods from the collection to operate on the collections. It has two implementations they are ArrayList and LinkedList. It can contain duplicate items also.

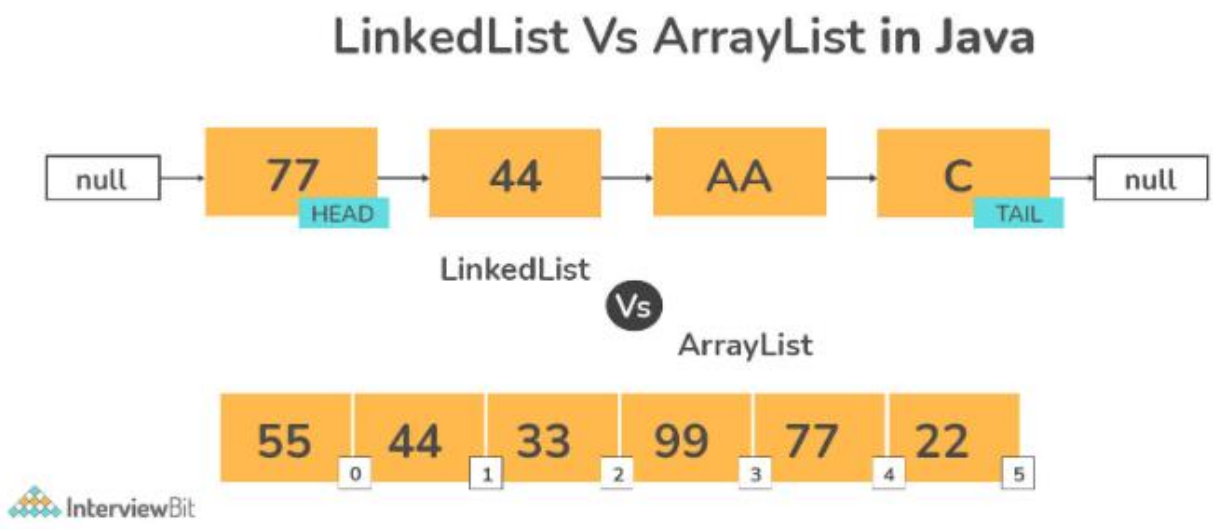
10. Explain Queue interface?

Queue is a data structure, besides basic collection operations queue provides additional insertion, removal, and inspection operations. Queue typically, but not necessarily orders elements in FIRST IN FIRST OUT (FIFO) manner.

11. Explain Map?

Map is an object that maps keys to values, it can't contain duplicate keys, one has map to at most one value. It has methods (put, get, remove, containsKey, containsValue, size, and empty) to perform basic operations on map data, for the bulk operations we use clear, putAll methods, for collection view keyset, entrySet, and values are used.

12. List out differences between ArrayList and LinkedList?



ArrayList	***LinkedList***
Elements of this class are stored in a dynamic array.	Elements of this class are stored in a doubly linked list.

This class implements List interface. As a result, it can be used as List.	This class implements both List and Deque as a result it can be used as both List and Deque.
Because of internal implementations, manipulating ArrayList takes longer time. Internally ArrayList is scanned, and memory are shifted whenever we remove an element.	There is no internal implementations on LinkedList, so it doesn't take much time to manipulate. The reference link is changed after traversing the list.
This class is more useful when the application requires the data storage and access.	This class is more useful when the application requires data manipulation.

13. Differentiate between ArratList and Vector?

ArrayList	***Vector***
Can't be synchronized	Can be synchronized
It is not a legacy class	It is a legacy class
Not thread safe	Thread safe

14. Difference between Iterator and ListIterator?

Iterator	***ListIterator***
Used to traverse array items in forward direction only	Used to traverse array items in both directions.
Can be used with Queue, List, Set	Used with List
Performs only remove operation	Can perform operations like add, remove, set operations while traversing the collection

15. Difference between List and Set?

List	***Set***
List maintains insertion order.	Set doesn't maintain insertion order.
Duplicate elements are allowed in list.	Duplicate elements are not allowed.
List is an index based.	Set is not an index based.

Elements can accessed by their index.	Set not allow access to their elements from their certain position.
It is possible to store null elements several times.	A null value can stored only once.

16. Differentiate between HashSet and TreeSet?

HashSet	***TreeSet***
HashSet doesn't provide any ordering guarantee.	TreeSet provides ordering or sorting guarantee.
HashSet uses equals() method for comparison and avoids duplicates.	TreeSet uses compareTo() method for comparison.
HashSet uses HashTable as data structure.	TreeSet uses Red-Black Tree.
HashSet allows one null element.	TreeSet doesn't allow null objects.
Implemented using HasMap.	Implemented using TreeMap.
HashSet is faster.	TreeSet is slower.

17. Can we add null element to the TreeSet?

No. We can't add null element to the TreeSet because it uses compareTo() method for comparing, it throws NullPointerException if it contains null element.