List and Set both are interfaces. They both extends Collection interface

The List interface allows storing the ordered collection. It is a child interface of Collection. It is an ordered collection of objects in which duplicate values are allowed to store. List preserves the insertion order, it allows positional access and insertion of elements.

The set interface in the java.util package and extends Collection interface is an unordered collection of objects in which duplicate values cannot be stored. It is an interface that implements the maths set. This interface contains the methods inherited from the Collection interface and adds a feature that restricts to insert the duplicate elements.

Example:
Input : Add Elements = [1, 2, 3, 1]
Output: Set = [1, 2, 3]
List = [1, 2, 3, 1]

Input : Add Elements = [a, b, d, b]
Output: Set = [a, b, d]
List = [a, b, d, b]

List	Set
The insertion order is maintained by the List.	It doesn't maintain the insertion order of elements.
Allows duplicate elements	Doesn’t allow duplicate elements
List allows us to add any number of null values.	Set allows us to add only one null value
The List implementation classes are LinkedList and ArrayList,vector and stack	The Set implementation classes are TreeSet, HashSet and LinkedHashSet.
Elements by their position can be accessed.	Position access to elements is not allowed.
ListIterator can be used to traverse a List in both the directions(forward and backward)	We can use Iterator (It works with List too) to traverse a Set.

```
// Implementation of List and Set in Java
import java.io.*;
import java.util.*;

class ListAndSet {
    public static void main(String[] args)
    {
        // List declaration
        List<Integer> lst = new ArrayList<>();
        lst.add(10);
        lst.add(45);
        lst.add(10);
        lst.add(46);
        lst.add(4);
```

```
                    lst.add(45);

                    // Set declaration
                    Set<Integer> st = new HashSet<>();
                    st.add(10);
                    st.add(45);
                    st.add(10);
                    st.add(46);
                    st.add(4);
                    st.add(45);

                    // printing list
                    System.out.println("List = " + lst);
                    // printing Set
                    System.out.println("Set = " + st);
                }
        }
```

OUTPUT:

```
C:\Users\Jeevan>java ListAndSet

List = [10, 45, 10, 46, 4, 45]

Set = [4, 10, 45, 46]
```

Advantages of Collection Framework:

Java Collection framework has several advantages, few of them include:
1. Consistent API: Java collection framework consists of a consistent API that has all the essential arrangement of interfaces like collections, Lists, Set, List, etc, and classes that implement the interfaces like ArrayList, Vector, etc.
2. Reduce Programming Exertion: The collection framework provides various provisions to implement an operation at an ease, this helps the programmer to focus on the main task, rather than the basic operations.
3. Increases Program Quality and Speed: The collection Framework helps to increase the execution speed and quality by reducing program size and letting the programmer handle data at an ease.
4. **Reduces effort to design new APIs:** This is the flip side of the previous advantage. Designers and implementers don't have to reinvent the wheel each time they create an API that relies on collections; instead, they can use standard collection interfaces.

Collection Hierarchy in Java:

The hierarchy of the entire collection framework consists of four core interfaces such as Collection, List, Set, Map, and two specialized interfaces named SortedSet and SortedMap for sorting.All the interfaces and classes for the collection framework are located in java.util package. The diagram of Java collection hierarchy is shown in the below figure.

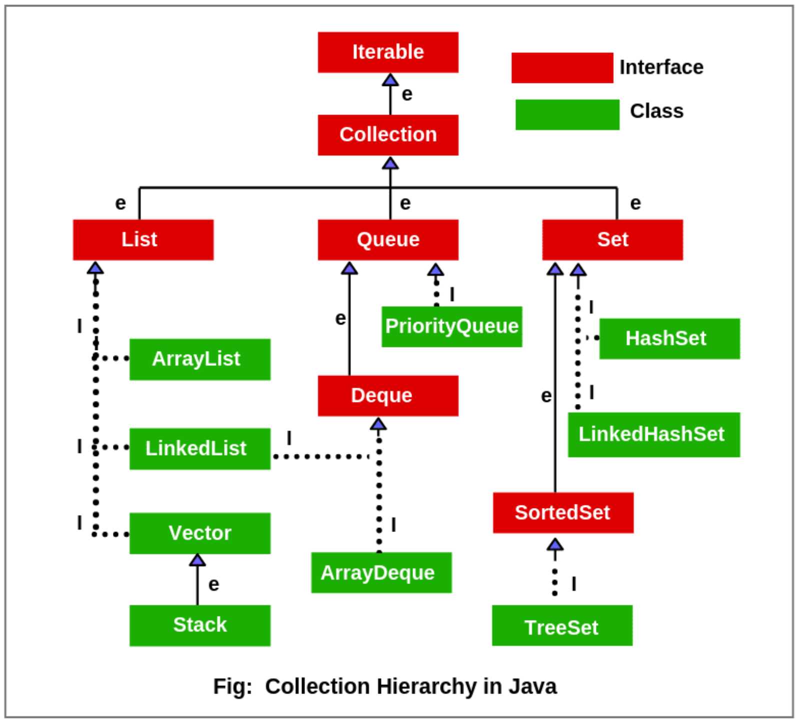

Fig: Collection Hierarchy in Java

e→ extends, I→ implements

Extends: Extends is a keyword that is used for developing inheritance between two classes and two interfaces.

Implements: Implements is a keyword used for developing inheritance between class and interface.

Collection Interface in Java

The basic interface of the collections framework is the Collection interface which is the root interface of all collections in the API (Application programming interface).

It is placed at the top of the collection hierarchy in java. It provides the basic operations for adding and removing elements in the collection.

The Collection interface extends the Iterable interface. The iterable interface has only one method called iterator(). The function of the iterator method is to return the iterator object. Using this iterator object, we can iterate over the elements of the collection.

List, Queue, and Set have three component which extends the Collection interface. A map is not inherited by Collection interface.

List Interface

This interface represents a collection of elements whose elements are arranged sequentially ordered.

List maintains an order of elements means the order is retained in which we add elements, and the same sequence we will get while retrieving elements.

We can insert elements into the list at any location. The list allows storing duplicate elements in Java.

ArrayList, vector, and LinkedList are three concrete subclasses that implement the list interface.

Set Interface

This interface represents a collection of elements that contains unique elements. i.e, It is used to store the collection of unique elements.

Set interface does not maintain any order while storing elements and while retrieving, we may not get the same order as we put elements. All the elements in a set can be in any order.

Set does not allow any duplicate elements.

HashSet, LinkedHashSet, TreeSet classes implements the set interface and sorted interface extends a set interface.

It can be iterated by using Iterator but cannot be iterated using ListIterator.

SortedSet Interface

This interface extends a set whose iterator transverse its elements according to their natural ordering.

TreeSet implements the sorted interface

Queue Interface

A queue is an ordered of the homogeneous group of elements in which new elements are added at one end(rear) and elements are removed from the other end(front). Just like a queue in a supermarket or any shop.

This interface represents a special type of list whose elements are removed only from the head.

LinkedList, Priority queue, ArrayQueue, Priority Blocking Queue, and Linked Blocking Queue are the concrete subclasses that implement the queue interface.

Deque Interface

A deque (double-ended queue) is a sub-interface of queue interface. It is usually pronounced “deck”.

This interface was added to the collection framework in Java SE 6.

Deque interface extends the queue interface and uses its method to implement deque. The hierarchy of the deque interface is shown in the below figure.

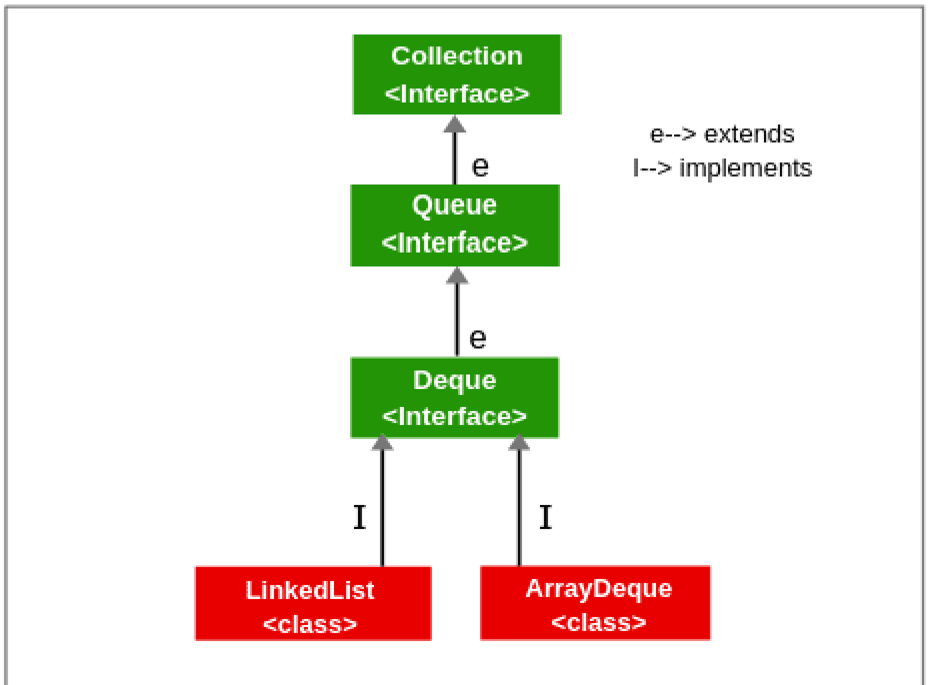

Fig: Deque Hierarchy

It is a linear collection of elements in which elements can be inserted and removed from either end. i.e, it supports insertion and removal at both ends of an object of a class that implements it.

LinkedList and ArrayDeque classes implement the Deque interface.

Map Interface

Map interface is not inherited by the collection interface. It represents an object that stores and retrieves elements in the form of a Key/Value pairs and their location within the Map are determined by a Key.

The hierarchy of the map interface is shown in the below figure.

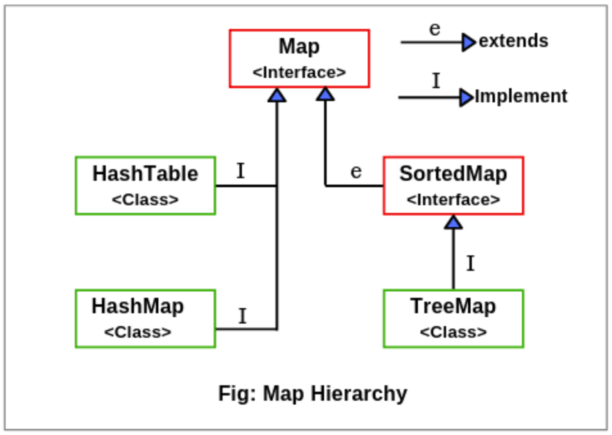

Fig: Map Hierarchy

Map uses a hashing technique for storing key-value pairs.

It doesn’t allow to store the duplicate keys but duplicate values are allowed.

HashMap, HashTable, LinkedHashMap, TreeMap classes implements Map interface.

SortedMap Interface

This interface represents a Map whose elements are stored in their natural ordering. It extends the Map interface which in turn is implemented by TreeMap classes.

Methods of Collection Interface in Java

The Collection interface consists of a total of fifteen methods for manipulating elements in the collection. They are as follows

1. **add():** This method is used to add or insert an element in the collection. The general syntax for add() method is as follow:

```
add(Object element) : boolean
```

If the element is added to the collection, it will return true otherwise false, if the element is already present and the collection doesn’t allow duplicates.

2. **addAll():** This method adds a collection of elements to the collection. It returns true if the elements are added otherwise returns false. The general syntax for this method is as follows:

```
addAll(Collection c) : boolean
```

3. **clear():** This method clears or removes all the elements from the collection. The general form of this method is as follows:

```
clear() : void
```

This method returns nothing.

4. **contains():** It checks that element is present or not in a collection. That is it is used to search an element. The general for contains() method is as follows:

```
contains(Object element) : boolean
```

This method returns true if the element is present otherwise false.

5. **containsAll():** This method checks that specified a collection of elements are present or not. It returns true if the calling collection contains all specified elements otherwise return false. The general syntax is as follows:

```
containsAll(Collection c) : boolean
```

6. **equals():** It checks for equality with another object. The general form is as follows:

```
equals(Object element) : boolean
```

7. **hashCode():** It returns the hash code number for the collection. Its return type is an integer. The general form for this method is:

```
hashCode() : int
```

8. **isEmpty():** It returns true if a collection is empty. That is, this method returns true if the collection contains no elements.

```
isEmpty() : boolean
```

9. **iterator():** It returns an iterator. The general form is given below:

```
iterator() : Iterator
```

10. **remove():** It removes a specified element from the collection. The general syntax is given below:

```
remove(Object element) : boolean
```

This method returns true if the element was removed. Otherwise, it returns false.

11. **removeAll():** The removeAll() method removes all elements from the collection. It returns true if all elements are removed otherwise returns false.

```
removeAll(Collection c) : boolean
```

12. **retainAll():** This method is used to remove all elements from the collection except the specified collection. It returns true if all the elements are removed otherwise returns false.

```
retainAll(Collection c) : boolean
```

13. **size():** The size() method returns the total number of elements in the collection. Its return type is an integer. The general syntax is given below:

```
size() : int
```

14. **toArray():** It returns the elements of a collection in the form of an array. The array elements are copies of the collection elements.

```
toArray() : Object[]
```

15. **Object[ ] toArray():** Returns an array that contains all the elements stored in the invoking collection. The array elements are the copies of the collection elements.

```
toArray(Object array[]) : Object[]
```