Collections Framework in Java

The java.util package contains all the classes and interfaces for Collection framework.

Map : An object that maps keys to values. A map is not ordered and cannot contain duplicate keys (but can contain duplicate values). Each key can map to at most one value.

Collection : A collection represents a group of objects, known as its elements. The JDK provides implementations of more specific subinterfaces like Set and List.

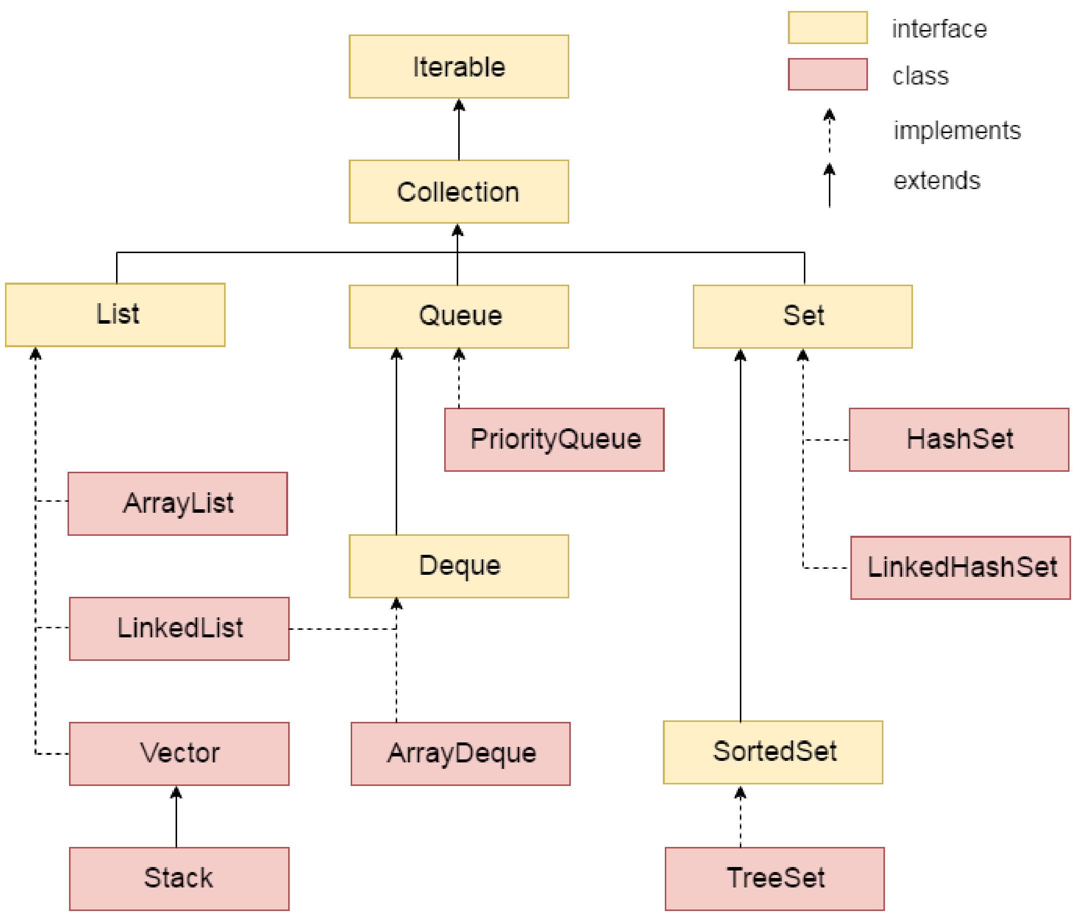
List : A list is an ordered list of objects, where the same object may well appear more than once. For example: [1, 7, 1, 3, 1, 1, 1, 5]. It makes sense to talk about the "third element" in a list. You can add an element anywhere in the list, change an element anywhere in the list, or remove an element from any position in the list.

Queue : A queue is also ordered, but you'll only ever touch elements at one end. All elements get inserted at the "end" and removed from the "beginning" (or head) of the queue. You can find out how many elements are in the queue, but you can't find out what, say, the "third" element is. You'll see it when you get there.

Set : A set is not ordered and cannot contain duplicates. Any given object either is or isn't in the set. {7, 5, 3, 1} is the exact same set as {1, 7, 3, 5}. You again can't ask for the "third" element or even the "first" element, since they are not in any particular order. You can add or remove elements, and you can find out if a certain element exists.

	List	Set	Queue	Map
Order	**Yes**	**No**	**Yes**	**No**
Duplicates	**Yes**	**No**	**Yes**	**No (Allow duplicate values not keys)**
Null Values	**Yes**	**Single Null**	**Yes (LinkedList Queue). No (Priority Queue).**	**Single null key and many null values**

Hierarchy of Collection Interface :



Methods of Collection Interface :

No.	Method	Description
1	public boolean add(Object element)	is used to insert an element in this collection.
2	public boolean addAll(Collection c)	is used to insert the specified collection elements in the invoking collection.
3	public boolean remove(Object element)	is used to delete an element from this collection.
4	public boolean removeAll(Collection c)	is used to delete all the elements of specified collection from the invoking collection.
5	public boolean retainAll(Collection c)	is used to delete all the elements of invoking collection except the specified collection.
6	public int size()	return the total number of elements in the collection.

7	public void clear()	removes the total no of element from the collection.
8	public boolean contains(Object element)	is used to search an element.
9	public boolean containsAll(Collection c)	is used to search the specified collection in this collection.
10	public Iterator iterator()	returns an iterator.
11	public Object[] toArray()	converts collection into array.
12	public boolean isEmpty()	checks if collection is empty.
13	public boolean equals(Object element)	matches two collection.
14	public int hashCode()	returns the hashcode number for collection.

Methods of Iterator Interface :

No.	Method	Description
1	public boolean hasNext()	It returns true if iterator has more elements.
2	public Object next()	It returns the element and moves the cursor pointer to the next element.
3	public void remove()	It removes the last elements returned by the iterator. It is rarely used.

ArrayList vs LinkedList :

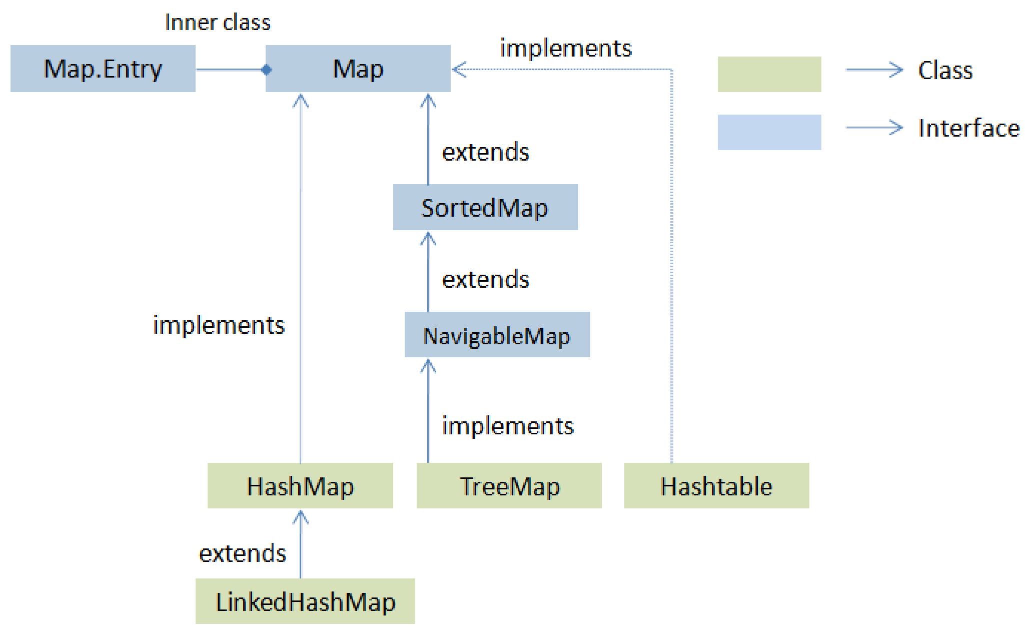
ArrayList	LinkedList
1) ArrayList internally uses **dynamic array** to store the elements.	LinkedList internally uses **doubly linked list** to store the elements.
2) Manipulation with ArrayList is **slow** because it internally uses array. If any element is removed from the array, all the bits are shifted in memory.	Manipulation with LinkedList is **faster** than ArrayList because it uses doubly linked list so no bit shifting is required in memory.
3) 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.

ArrayList vs Vector :

ArrayList	Vector
1) ArrayList is **not synchronized**.	Vector is **synchronized**.
2) ArrayList **increments 50%** of current array size if number of element exceeds from its capacity.	Vector **increments 100%** means doubles the array size if total number of element exceeds than its capacity.
3) ArrayList is **not a legacy** class, it is introduced in JDK 1.2.	Vector is a **legacy** class.
4) ArrayList is **fast** because it is non-synchronized.	Vector is **slow** because it is synchronized i.e. in multithreading environment, it will hold the other threads in runnable or non-runnable state until current thread releases the lock of object.
5) ArrayList uses **Iterator** interface to traverse the elements.	Vector uses **Enumeration** interface to traverse the elements. But it can use Iterator also.

Hierarchy of Map Interface :



Useful Methods of Map Interface :

Method	Description
Object put(Object key, Object value)	It is used to insert an entry in this map.
void putAll(Map map)	It is used to insert the specified map in this map.
Object remove(Object key)	It is used to delete an entry for the specified key.
Object get(Object key)	It is used to return the value for the specified key.
boolean containsKey(Object key)	It is used to search the specified key from this map.
Set keySet()	It is used to return the Set view containing all the keys.
Set entrySet()	It is used to return the Set view containing all the keys and values.

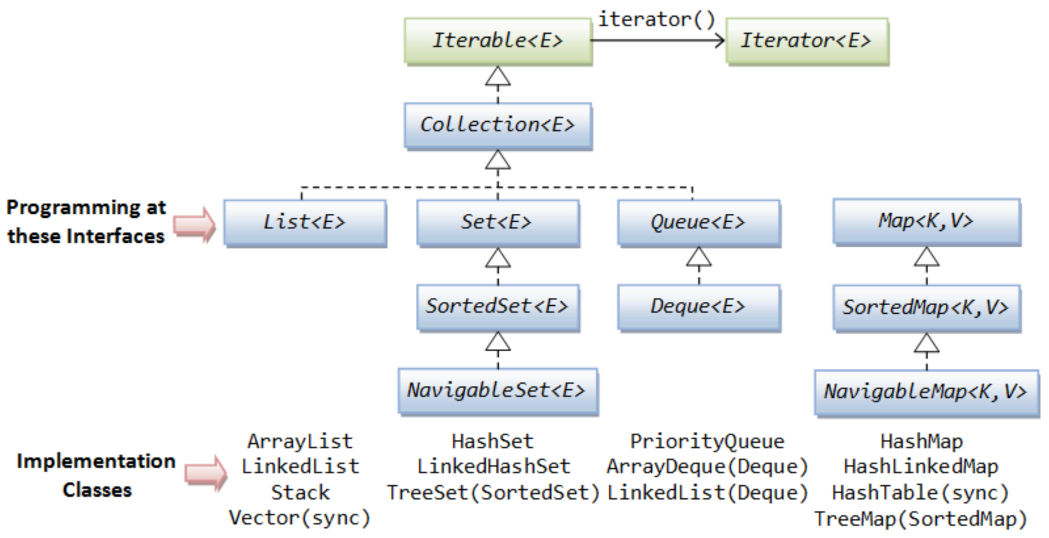
Methods of Map.Entry Interface :

Method	Description
Object getKey()	It is used to obtain key.
Object getValue()	It is used to obtain value.

HashMap vs HashTable :

HashMap	Hashtable
1) HashMap is **non synchronized**. It is not-thread safe and can't be shared between many threads without proper synchronization code.	Hashtable is **synchronized**. It is thread-safe and can be shared with many threads.
2) HashMap **allows one null key and multiple null values**.	Hashtable **doesn't allow any null key or value**.
3) HashMap is a **new class introduced in JDK 1.2**.	Hashtable is a **legacy class**.
4) HashMap is **fast**.	Hashtable is **slow**.
5) We can make the HashMap as synchronized by calling this code Map m = Collections.synchronizedMap(hashMap);	Hashtable is internally synchronized and can't be unsynchronized.
6) HashMap is **traversed by Iterator**.	Hashtable is **traversed by Enumerator and Iterator**.
7) Iterator in HashMap is **fail-fast**.	Enumerator in Hashtable is **not fail-fast**.
8) HashMap inherits **AbstractMap** class.	Hashtable inherits **Dictionary** class.

Collections Framework Implementation Classes Summary :



Comparable vs Comparator Interfaces :

Comparable	Comparator
1) Comparable provides **single sorting sequence**. In other words, we can sort the collection on the basis of single element such as id or name or price etc.	Comparator provides **multiple sorting sequence**. In other words, we can sort the collection on the basis of multiple elements such as id, name and price etc.
2) Comparable **affects the original class** i.e. actual class is modified.	Comparator **doesn't affect the original class** i.e. actual class is not modified.
3) Comparable provides **compareTo() method** to sort elements.	Comparator provides **compare() method** to sort elements.
4) Comparable is found in **java.lang** package.	Comparator is found in **java.util** package.
5) We can sort the list elements of Comparable type by **Collections.sort(List)** method.	We can sort the list elements of Comparator type by **Collections.sort(List,Comparator)** method.

Legacy Data Structures in Java

Legacy classes and interfaces are the classes and interfaces that formed the collections framework in the earlier versions of Java and has now been restructured or re-engineered. They are fully compatible with the framework.

All legacy classes were re-engineered to support generic in JDK5.

Legacy = heritage of old java version.

Legacy classes and interfaces - Enumeration, Vector, Stack, Dictionary, HashTable, Properties...