Java Collections

Revision Notes

Created by: Vishal Arya

Collections in Java

Collection - collection allows a group of objects to be treated as a single unit.

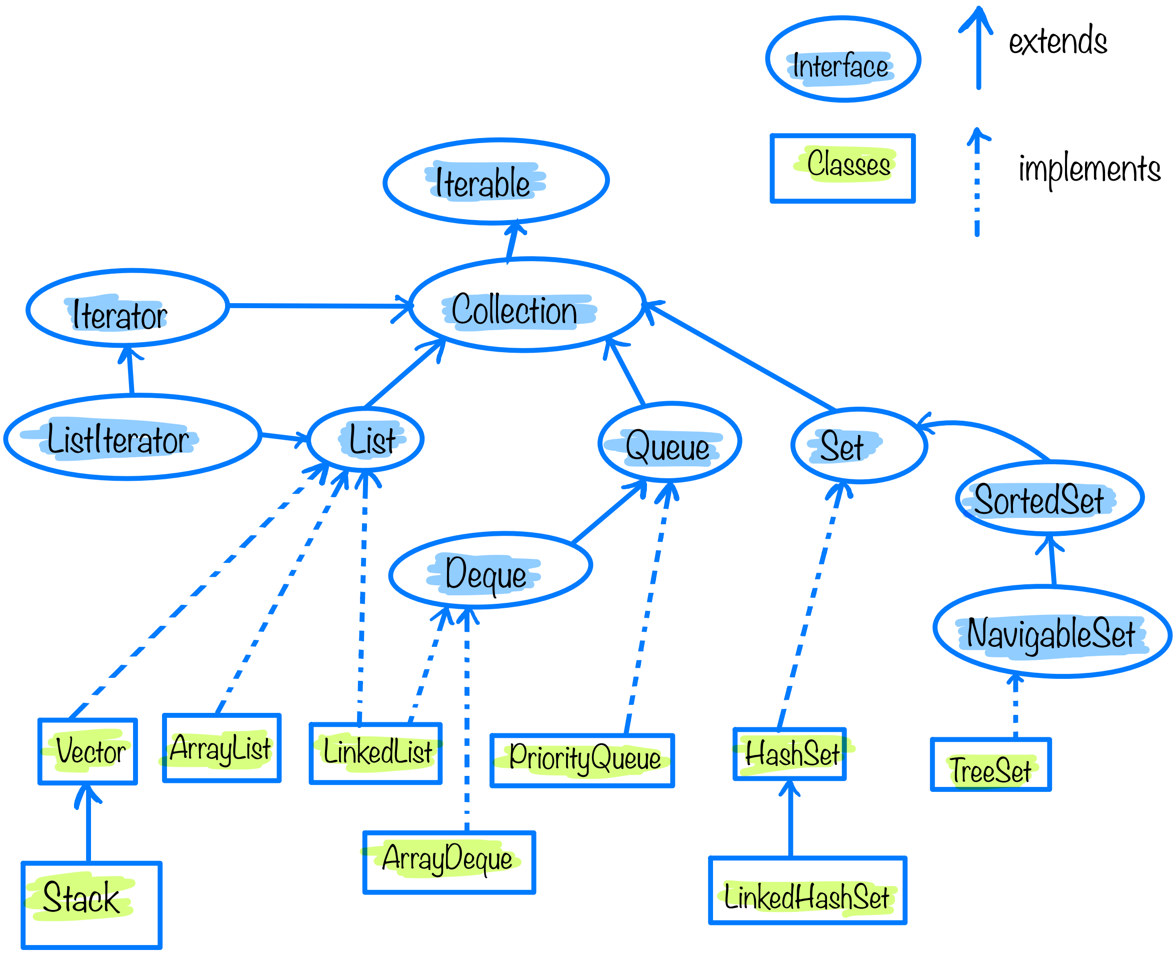
Collections framework in Java provides a set of standard utility classes and interfaces for managing the different kinds of collections.

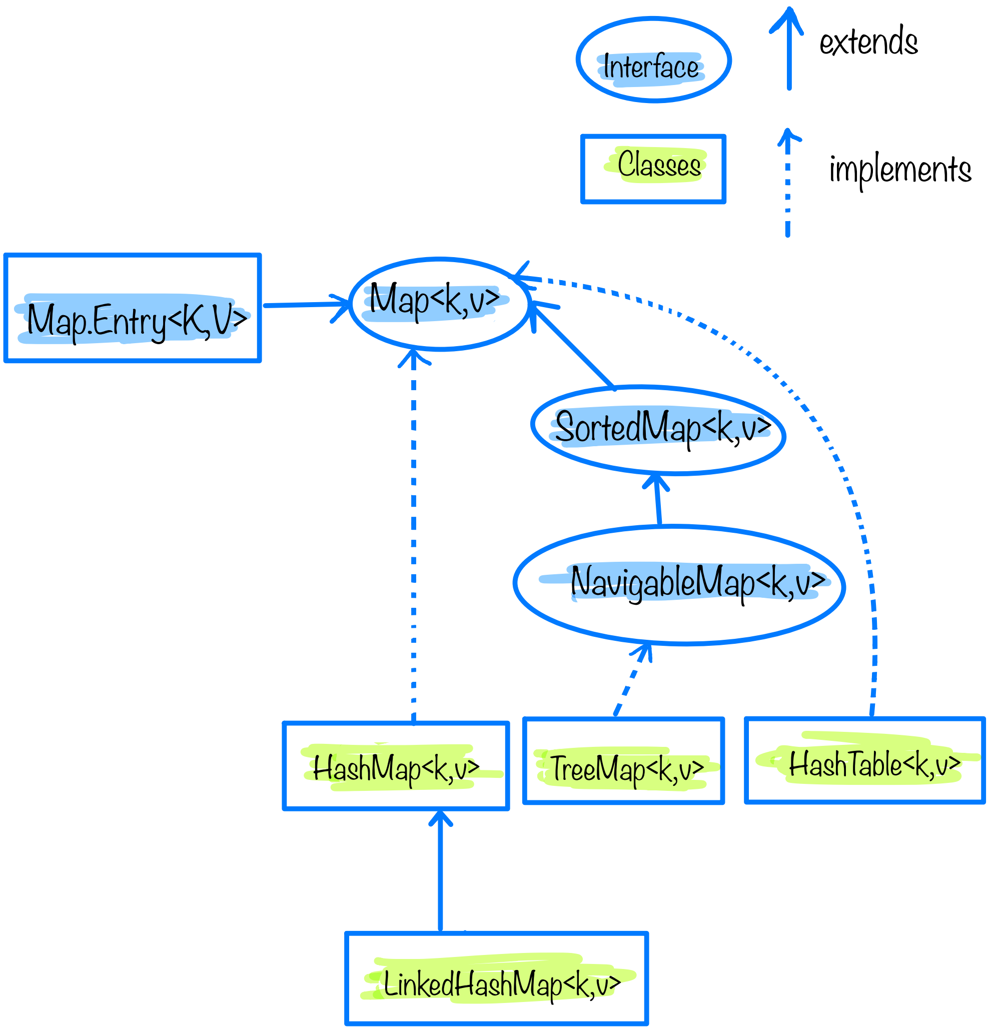
java.util package contains all the classes and interfaces

Three main parts:

- core interfaces
- implementation of interfaces (concrete classes)
- Static methods like sorting,searching in the Colllections and Arrays

Collection interface extends the Iterable interface that specifies an iterator to sequentially access the elements of an Iterable object.


Interface
extends
Classes
implements
Iterable
Iterator
Collection
ListIterator
List
Queue
Set
SortedSet
Deque
NavigableSet
Vector
ArrayList
LinkedList
PriorityQueue
HashSet
TreeSet
Stack
ArrayDeque
LinkedHashSet


Interface
extends
Classes
implements
Map.Entry<K,V>
Map<k,v>
SortedMap<k,v>
NavigableMap<k,v>
HashMap<k,v>
TreeMap<k,v>
HashTable<k,v>
LinkedHashMap<k,v>

Collection Interface

boolean containsAll(Collection<?> c)

boolean addAll(Collection<? extends E> c)

boolean removeAll(Collection<?> c)

boolean retainAll(Collection<?> c)

boolean clear()

Lists

Order - maintain

Can contain duplicate

Indexing starts from 0

Implementation of list

ArrayList

LinkedList

Vector

ArrayList

Dynamic array

Internally uses normal array with default capacity

When capacity is reached it will create a new array of bigger size and copies. All the elements to the new array.

New array reference is used as the old array is no longer in use. It is garbage collected in the next garbage collection.

Vector

It a legacy class.

Dynamic in nature .

Position based access.

ArrayList and vector classes offers comparable performance, but a vector suffers a slight performance penalty due to synchronisation.

LinkedList

Its implementation uses a doubly linked list.

Insertion and deletion are very efficient in doubly linked list.

	Null elements	Duplicate	Order maintain	Thread safe
ArrayList	✓	✓	Insertion order	✗
LinkedList	✓	✓	Insertion order	✗
Vector	✓	✓	Insertion order	✓

ListIterator

Queue

FIFO - First in first out

Implementation

Deque

ArrayDeque

PriorityQueue

Deque

Deque interface extends Queue interface to allow double-ended queue.

It allows operation not just at its head, but also at its tail.

Element can be removed or inserted from either end .

Deque can be used as FIFO queue, where elements added at the tail are presented at the head for inspection or removal in the same order thus implementing FIFO order.

Generally used:

In case of stack implementation then go for Stack class.

In case of normal queue then go for LinkedList class.

In case of double-ended queue then go ArrayDeque class.

PriorityQueue

PriorityQueue works on priority.

The implementation is based on the priority heap.

In case of several element having the same priority, one of them is chosen arbitrarily.

Element of a PriorityQueue are not sorted.

The queue only guarantee is that element can be removed in a priority order, and any traversal using an iterator does not.

	Null elements	Duplicate	Order maintain	Thread safe
PriorityQueue	✗	✓	Natural order or according to supplied comparator	✗
ArrayDeque	✗	✓	Supports both FIFO and LIFO	✗

Sets

Set is a linear collection of object with no duplicate.

Implementation

HashSet

LinkedHashSet

TreeSet

SortedSet

The SortedSet interface extends Set interface.
Elements are sorted in nature . Traverse the set either using for loop or an iterator will access the element according to the ordering.

NavigableSet

NavigableSet set extends the SortedSet interface with navigation(Search methods) method to find the closest matches for a specific search target.

	Null elements	Duplicate	Order maintain	Thread safe
HashSet	Maximum one	✗	✗	✗
LinkedHashSet	Maximum one	✗	Insertion order	✗
TreeSet	✗	✗	Natural order or according to supplied comparator	✗

Map

Map defines mapping from keys to values.

Pair of <key, value> called an entry.

Keys should be unique.

Mappings can be viewed as a collection in various way: a key set, a value set, entry set.

Implementation

HashMap - implement unordered map

HashTable - implement unordered map

LinkedHashMap - implement ordered map

TreeMap - implement SortedMap

SortedMap

SortedMap extends the Map interface to provide the functionality for implementing maps with sorted keys.

NavigableMap

NavigableMap interface extends the SortedMap interface with navigation method to find the closest matches for a specific search target.

TreeMap

By default, operations on sorted maps rely on the natural ordering of the keys.
Custom ordering can be specified by passing a customized comparator to the constructor .
Behind the scene, it uses balanced tree which deliver excellent performance for all the operation.
However searching in HashMap can be faster than in a TreeMap, as hashing algorithms usually offer better performance than the search algorithm for balanced trees.

	Null elements	Duplicate	Order maintain	Thread safe
HashMap	Maximum one	Doesn't allow duplicate key, can have duplicate values	✗	✗
LinkedHashMap	Maximum one	Doesn't allow duplicate key, can have duplicate values	Insertion order	✗
TreeMap	Doesn't allow even a single key but can have multiple null values	Doesn't allow duplicate key, can have duplicate values	Natural order or according to supplied comparator	✗