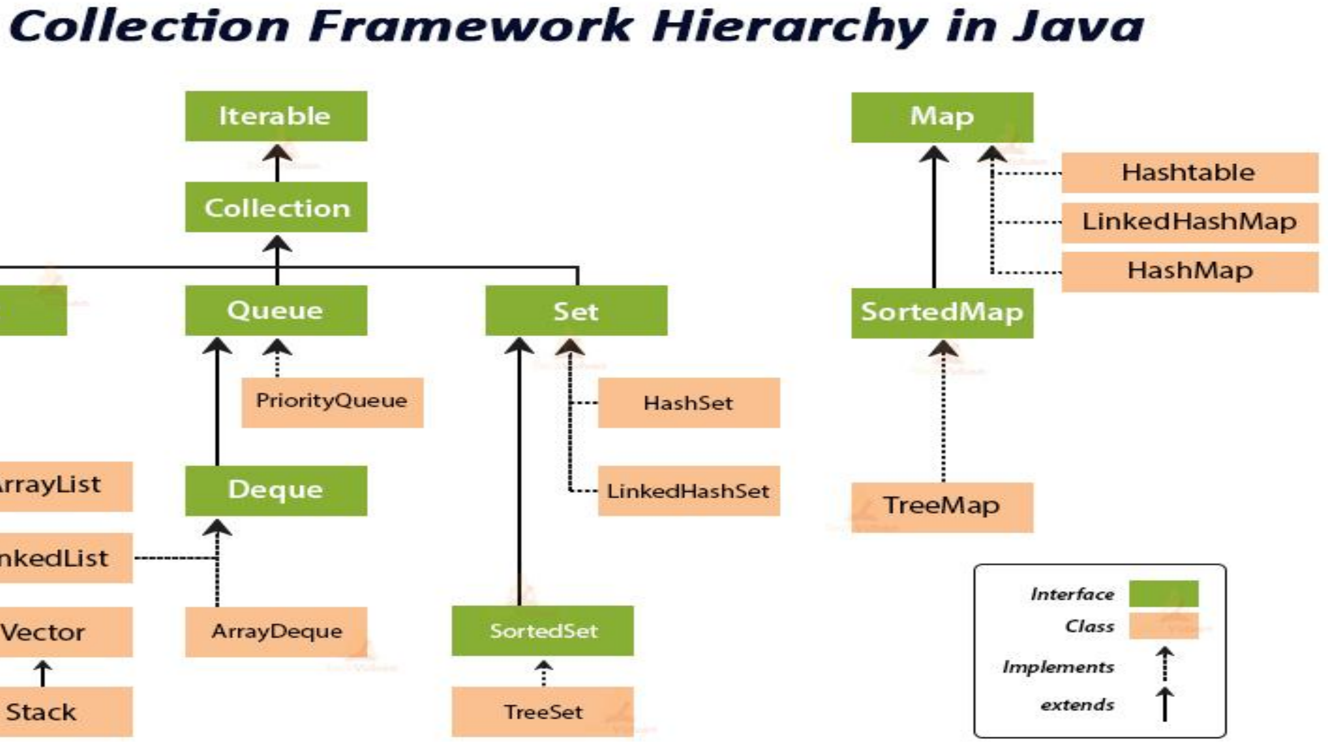
Collection Framework

Collection Framework

- A collection object or container object is an object which can store a group of other objects.
- A collection object has a class called as ‘collection class’ or ‘container class’.
- All the collection classes are available in the package java.util. Util stands for utility.
- A group of collection classes is called a collection framework or we can say a collection framework is a class library to handle group of objects.
- A collection object stores references of other objects.
- **Collections store only objects and not primitive type data**.

Collection Framework



Collection Framework

Collection Interface

The Collection interface is the root interface and provides common methods like add, remove, clear, contains, equals, hashcode, and iterator.

List interface

The List interface extends the Collections interface. The List represents index based ordered collection of the objects. The elements contained in the List are ordered and can be inserted, accessed or searched based on their index. The list may contain duplicate elements. The main classes implementing the List interface are ArrayList and LinkedList.

Collection Framework

Set interface

The Set interface extends the Collection interface and represents a collection that does not contain any duplicate elements (it can only have one null element as well). The main classes implementing the Set interface are TreeSet, HashSet and LinkedHashSet.

Queue Interface

The Queue interface extends Collection interface and represents a collection that is usually ordered by FIFO (first in first out) order. The main classes implementing the Queue interface are ArrayDeque and PriorityQueue.

Collection Framework

Map Interface

The map interface is a root interface and allows storing the key value pairs. The map does not allow duplicate keys. Map interface does not guarantee the order of the elements, however, some implementations like TreeMap does. The main classes implementing the Map interface are Hashtable, HashMap, TreeMap and LinkedHashMap.

Collection Framework

Utility Classes

In addition to the above mentioned main interfaces and classes, there are two utility classes that are part of the Java collection framework.

1. Collections class

The Collections class contains static utility methods that either accepts or returns the collection. The collection class provides many useful methods for shuffling, reversing, sorting and searching collection objects.

2. Arrays Class

Similar to the Collections class, the Arrays class contains static utility methods for manipulating arrays. The Arrays class provides many useful methods for sorting, searching, copying and filling the arrays.

Collection Framework

Retrieving Elements From Collections

Four ways to retrieve elements from a collection object

- Using for-each loop
- Using Iterator interface
- Using ListIterator interface
- Using Enumeration interface

Collection Framework

Retrieving Elements From Collections : Using for-each loop

Syntax :

```
for(variable: collection-object)
{
        statements;
}
```

for-each can be applied to cycle through any collection of objects that implement the Iterable interface.

for-each can be used for array iteration also.

Collection Framework

Retrieving Elements From Collections : Using Iterator
Each of the collection class (not Map classes) provides an iterator( ) method that returns an iterator to the start of the collection.

By using this Iterator object, we can access each element in the collection, one element at a time.
Using Iterator we can iterate those containers only which implements Iterable interface.

Syntax:
interface Iterator <E>
Where E specifies the type of objects being iterated.

Collection Framework

Retrieving Elements From Collections :Iterator Interface

Iterator is an interface that contains methods to retrieve the elements one by one form a collection object. It has 3 methods:

- boolean hasNext( ) : Returns true if the iterator has more elements.
- element next( ) : Returns the next element in the iterator.
- void remove ( ) : Removes the last element returned by the iterator from the collection.

To get the iterator object **iterator()** method is used.

E.g. Iterator it = myArrayList.iterator();

Collection Framework

Retrieving Elements From Collections :ListIterator Interface

Syntax: interface ListIterator <E> , where E is type of elements being iterated.

To retrieve the elements from a collection object, both in forward and reverse directions. It has the following important methods:

- boolean hasNext( ) : Returns true if the ListIterator has more elements when traversing the list in forward direction.
- element next( ) : Returns the next element in the list.
- boolean hasPrevious( ) : Returns true if the iterator has more elements when traversing in reverse direction.
- element previous( ) : Returns the previous element in the list.
- void remove ( ) : Removes the last element returned by the next ( ) or previous ( ) methods.
- To get the ListIterator object **listIterator()** method is used.
  E.g. ListIterator it = myArrayList.listIterator();

Collection Framework

Retrieving Elements From Collections :Enumeration Interface

To retrieve elements one by one like Iterator. It has two methods:

- boolean hasMoreElements() : Checks if the Enumeration has any more elements or not.
- element nextElement() : Returns the next element that is available in Enumeration.

Unlike Iterator Enumertaion does not have an option to remove elements. So, Iterator is preferred to Enumeration.

- To get the Enumeration object **elements() or enumeration()** method is used.
  Enumeration em = Collections.enumeration(myArrayList);

Collection Framework

Collection Interface Methods

Collection is a generic interface that has this declaration: ***interface Collection <E>***

boolean add (E obj)	Add object to the invoking collection.
boolean addAll (Collection <? Extends E>	Adds all the elements of c to the invoking collection.
void clear ( )	Removes all elements from the invoking collection.
boolean contains (Object obj)	Returns true if obj is in invoking collection.
boolean containsAll (Collection <?> c)	Returns true if the invoking collection contains all elements of c.
boolean equals (Object obj)	Returns true if the invoking collection and obj are equal

Collection Framework

Collection Interface Methods

boolean isEmpty ( )	Returns true if the invoking collection is empty
Iterator<E> iterator ( )	Returns an iterator for the invoking collection.
boolean remove (Object obj)	Removes object from the invoking collection.
boolean removeAll (Collection <?> c)	Removes all elements of c from the invoking collection.
boolean retainAll (Collection <?> c)	Removes all elements from collection except those in c.
int size ( )	Returns the number of elements held in the invoking collection.
Object [ ] toArray( )	Returns an array that contains all the elements stored in the invoking collection.

Collection Framework

List Interface Methods : interface List <E>

All methods of Collection interface will be available for List also. Apart from that the following are the methods in the interface List <E>

void add (int index, E obj)	**Inserts** obj at the specified index.
void addAll (int index, Collection <? extends E> c)	Inserts all elements of c at the specified index.
E get (int index)	Returns element stored at specified index.
int indexOf (Object obj)	Returns the index of the first instance of obj in the invoking list. If it is not present, -1 is returned.
int lastIndexOf (Object obj)	Returns the index of the last instance of obj in the invoking list. If it is not present, -1 is returned.
ListIterator <E> listIterator( )	Returns a ListIterator to the start for the invoking list.

Collection Framework

List Interface Methods : interface List <E>

ListIterator <E> listIterator(int index)	Returns a ListIterator for the invoking list that begins at the specified index.
E remove (int index)	Removes an element at a particular index.
E set (int index, E obj)	Assigns obj to the location specified by index within the invoking list.
List<E> subList (int start, int end)	Returns a sublist. **Elements in the sublist are also referenced by the invoking object.**

Collection Framework

Set Interface Methods : interface Set <E>

It extends Collection and declares the behavior of a collection that does not allow duplicate elements. **It does not define any additional methods of its own**.

Collection Framework

SortedSet Interface Methods : interface SortedSet <E>

It extends Set interface. All methods of Collection interface will be available for SortedSet also. Apart from that the following are the methods in the interface SortedSet <E>

E first ( )	To obtain the first element in the set.
E last ( )	To obtain the last element in the set.
SortedSet<E> subSet(E fromElement, E toElement)	Returns a view of the portion of this set whose elements range from fromElement, inclusive, to toElement, exclusive. (If fromElement and toElement are equal, the returned set is empty.) The returned set is backed by this set, so changes in the returned set are reflected in this set, and vice-versa.

Collection Framework

SortedSet Interface Methods : interface SortedSet <E>

SortedSet<E> headSet(E toElement)	Returns a view of the portion of this set whose elements are strictly less than toElement. The returned set is backed by this set, so changes in the returned set are reflected in this set, and vice-versa.
SortedSet<E> tailSet(E fromElement)	Returns a view of the portion of this set whose elements are greater than or equal to fromElement. The returned set is backed by this set, so changes in the returned set are reflected in this set, and vice-versa.
Comparator<? super E> comparator()	Returns the comparator used to order the elements in this set, or null if this set uses the natural ordering of its elements.

Collection Framework

Queue Interface Methods : interface Queue <E>

It extends Set interface. All methods of Collection interface will be available for SortedSet also. Apart from that the following are the methods in the interface SortedSet <E>

boolean add(E e)	Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions, returning true upon success and throwing an IllegalStateException if no space is currently available.
boolean offer(E e)	Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions, returning true if element added false otherwise. When using a capacity-restricted queue, this method is generally preferable to add(E), which can fail to insert an element only by throwing an exception.

Collection Framework

Queue Interface Methods : interface Queue <E>

E poll()	Retrieves and removes the head of this queue, or returns null if this queue is empty.
E remove()	Retrieves and removes the head of this queue. This method differs from poll only in that it throws an exception if this queue is empty.
E peek()	Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.
E element()	Retrieves, but does not remove, the head of this queue. This method differs from peek only in that it throws an exception if this queue is empty.

Collection Framework

Interfaces for Map

The following interfaces supports map:

Interface	Description
Map	Maps unique keys to values
Map.Entry	Describes an element (a key/value pair).
SortedMap	Extends Map so that the keys are maintained in ascending order.

Collection Framework

The Map Interface : interface Map <K,V>

- Although part of the Collections Framework, maps are not, themselves, collections because they do not implement the Collection interface. However we can obtain a collection-view of a map.
- To get a collection-view of a map, we can use entrySet( ) method which returns a Set that contains the elements in the map.
- To obtain a collection-view of the values, we can use values ( ).
- Collection-views are the means by which maps are integrated into the larger Collections Framework.

Collection Framework

The SortedMap Interface : interface SortedMap <K,V>

- It ensures that the entries are maintained in ascending order based on the keys.

The Map.Entry Interface : interface Map.Entry <K,V>

- We know entrySet( ) method of Map interface returns a Set containing the map entries. Each of these set elements is a Map.Entry object.

Collection Framework

Map Interface Methods : interface Map <K,V>

E clear ( )	Removes all key/value pairs from the invoking map.
boolean containsKey(Object k)	Checks whether the invoking map contains k as key or not.
int size()	Returns the number of key-value mappings in this map
boolean isEmpty()	Returns true if this map contains no key-value mappings.
V get(Object key)	Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.

Collection Framework

Map Interface Methods : interface Map <K,V>

V put(K key, V value)	Associates the specified value with the specified key in this map (optional operation). If the map previously contained a mapping for the key, the old value is replaced by the specified value.
V remove(Object key)	Removes the mapping for a key from this map if it is present.
Set<K> keySet()	Returns a Set view of the keys contained in this map. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa.

Collection Framework

Map Interface Methods : interface Map <K,V>

Collection<V> values()	Returns a Collection view of the values contained in this map. The collection is backed by the map, so changes to the map are reflected in the collection, and vice-versa.
Set<Map.Entry<K,V>> entrySet()	Returns a Set view of the mappings contained in this map. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa.

Collection Framework

SortedMap Interface Methods : interface SortedMap <K,V>

Apart from all the methods of interface Map, it contains the following methods

Comparator<? super K> comparator()	Returns the comparator used to order the keys in this map, or null if this map uses the natural ordering of its keys.
SortedMap<K,V> headMap(K toKey)	Returns a view of the portion of this map whose keys are strictly less than toKey.
SortedMap<K,V> tailMap(K fromKey)	Returns a view of the portion of this map whose keys are greater than or equal to fromKey.

Collection Framework

SortedMap Interface Methods : interface SortedMap <K,V>

K firstKey()	Returns the first (lowest) key currently in this map.
K lastKey()	Returns the last (highest) key currently in this map.

Collection Framework

Map.Entry Interface Methods : interface Map.Entry <K,V>

Apart from all the methods of interface Map, it contains the following methods

`K getKey()`	Returns the key corresponding to this entry.
`V getValue()`	Returns the value corresponding to this entry.
`V setValue(V value)`	Replaces the value corresponding to this entry with the specified value

Collection Framework

Collection Classes : HashSet Class

- A HashSet represents a set of elements (objects).
- It does not guarantee the order of elements.
- It does not allow duplicate elements to be stored.
- The implementation of HashSet is not synchronize. In case of multithreading, it must be externally synchronized.

  Syntax: HashSet <T>
  Here,<T> represents the generic type parameter that represents which type of elements (objects) are being stored into the HashSet.
  Ex. HashSet to store group of Strings (objects)
  HashSet <Strings> hs = new HashSet<Strings> ( )

Collection Framework

Collection Classes : HashSet Class

Constructors:

HashSet ( )

HashSet (int capacity)

HashSet (int capapcity, float loadfactor)

- Loadfactor determines the point where the capacity of HashSet would be incremented internally.
- Ex. if capacity = 100 and loadfactor = 0.5 , this means after storing the (101 x 0.5 = 50.5) 50th element into the HashSet its capacity internally be increased.
- **Creating a synchronized set**

  public static <T> Set<T> synchronizedSet(Set<T> s)

  Ex. Set synchSet = Collections.synchronizedSet(hs);

Collection Framework

Collection Classes : LinkedHashSet Class

- This is a subclass of HashSet class and does not contain any additional member of its own.
- It internally uses a linked list to store the elements.
- It maintains the insertion order.
- **Syntax: LinkedHashSet <T>**
  Here,<T> represents the generic type parameter that represents which type of elements (objects) are being stored into the HashSet.
  Ex. LinkedHashSet to store group of Strings (objects)
  LinkedHashSet <Strings> hs = new LinkedHashSet<Strings> ( )

Collection Framework

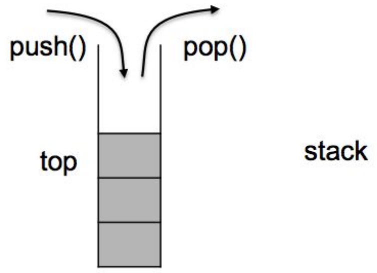
Collection Classes : TreeSet Class

- It creates a collection that uses a tree for storage.
- Objects are stored, in sorted, ascending order.
- Access and retrieval time is quite fast, which makes TreeSet an excellent choice when storing large amounts of sorted information that must be found quickly.

- **Syntax: class TreeSet <T>**
  Here,<T> represents the generic type parameter that represents which type of elements (objects) are being stored into the HashSet.
  Ex. TreeSet to store group of Strings (objects)
  TreeSet <Strings> hs = new TreeSet<Strings> ( )

Collection Framework

Collection Classes : Stack Class



A stack represents a group of elements stored in LIFO (Last In First Out) order.

Ex. A pile of plates, CD disk holder.

Stack is synchronized.

Syntax : class Stack <E>

Stack <Integer> obj = new Stack <Integer>();

Methods:

boolean empty( ) : Checks whether stack is empty or not.

element peek( ) : Returns top without removing it.

element pop ( ) : Pops (removes) the top element and returns it.

element push (element obj) : Pushes (adds) element to the stack.

int search (element obj)   // Returns -1 in case it is not found otherwise returns the position of the element.

Collection Framework

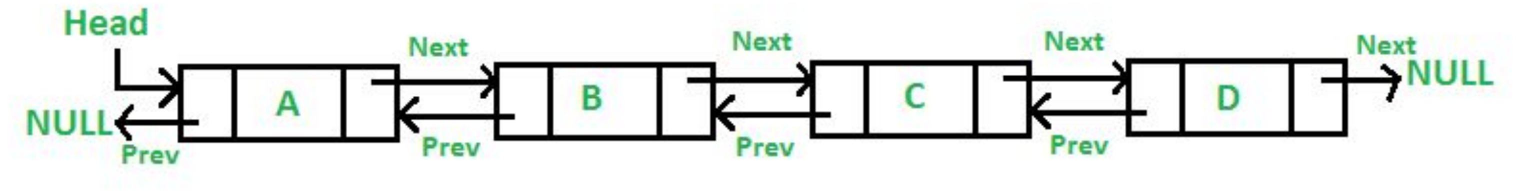
Collection Classes : LinkedList Class

A Linked list contains group of elements in the form of nodes. Each node will have data and link fields. Link fields contains the link to the next and previous nodes.

Syntax: class LinkedList<E>

Ex. LinkedList <String> linkedList = new LinkedList<String>()

Methods: See List interface methods.



Collection Framework

Collection Classes : LinkedList Class

Creating a synchronized list

```
public static <T> List<T> synchronizedList(List<T> list)
```

Ex. `List<String> synchList = Collections.synchronizedList(linkedList);`

Collection Framework

Collection Classes : ArrayList Class

An ArrayList is like an array which grows dynamically.

ArrayList is not synchronized.

Syntax: class ArrayList<E>

Ex. ArrayList <String> arList = new ArrayList<String> ( )

Methods: Please refer to the methods of List interface.

Collection Framework

Collection Classes : Vector Class

A Vector is like an array which grows dynamically.

Unlike ArrayList, Vector is **synchronized**.

Syntax: class Vector<E>

Ex. Vector <String> v = new Vector<String> ( )

Methods: Please refer to the methods of List interface.

Collection Framework

Map Classes : HashMap Class

It stores elements in the key-value pairs.

Keys must be unique.

It is not synchronized.

Syntax: class HashMap<K,V>

Ex. HashMap <String, Integer> hm = new HashMap<String, Integer> ( );

hm.put (“Mohit”,23);

hm.get(“Mohit”); // will return 23

The default initial capcity is 16 and the loadfactor is 0.75. 16x0.75 = 12 i.e. after storing 12th element, its capacity will become 32.

Collection Framework

Map Classes : HashMap Class

Methods: Please refer to the methods of Map interface.

Creating a synchronized map:

```
public static <K, V> Map<K, V> synchronizedMap(Map<K, V> m)
```

Ex. Map <String, Integer> synchMap = Collections.synchronizedMap(hm);

Collection Framework

Map Classes : HashTable Class

Similar to Hashmap, it also stores elements in the key-value pairs.
Keys must be unique.
Unlike HashMap, It is **synchronized**.

Syntax: class HashTable<K,V>
Ex. HashTable <String, Integer> ht = new HashTable<String, Integer> ( );
ht.put (“Mohit”,23);
ht.get(“Mohit”);   // will return 23

The default initial capcity is 16 and the loadfactor is 0.75. 16x0.75 = 12 i.e. after storing 12th element, its capacity will become 32.

Methods: Please refer to the methods of Map interface.
Enumeration <K> keys ( ) method : Returns an Enumeration of keys present in the HashTable

Collection Framework

Map Classes : TreeMap Class

It stores elements in the key-value pairs.

Keys must be unique.

It creates maps stored in a tree structure.

It is an efficient means of storing key-value pairs in sorted order and allows rapid retrieval.

Unlike HashMap, a TreeMap guarantees that its elements will be sorted in ascending key order.

Syntax: class TreeMap<K,V>

Ex. TreeMap <String, Integer> tm = new TreeMap<String, Integer> ( );

tm.put (“Mohit”,23);

tm.get(“Mohit”);   // will return 23

Methods: Please refer to the methods of Map interface.

Collection Framework

Utility Class: Arrays

Arrays class provides methods to perform certain operations on any one dimensional array.
All the methods of the Arrays class are static, so they can be called using Arrays.methodName( ).

Some Methods:

static void sort (array) : Sorting in ascending order. In place sorting of the array. It internally uses **QuickSort** algorithm.
static void sort(array, int start, int end) : start index inclusive whereas end index exclusive.
static int binarySearch(array, element) : For binary search, array must be sorted first.
static boolean equals(array1, array2)
static void fill(array,value)
static String toString(array) : Returns a String representation of the contents of the specified array.
Static T[] copyOf(originalarr, newarrlength) : Copies one array to the new array
static <T> List<T> asList(T... a) : Returns a fixed-size list backed by the specified array.

Collection Framework

Utility Class: Collections

The Collection Framework defines several algorithims that can e applied to collections and maps.
These algorithms are defined as static methods within the Collections class.

Some methods:

```
void shuffle (List <T> list )
void sort (List <T> list)                    // <T extends Comparable>
void sort (List <T> list, Comparator  comp)
T max (Collection <? extends T> c)         // <T extends Comparable>
T max (Collection <? extends T> c, Comparator comp)
T min (Collection <? extends T> c)          // <T extends Comparable>
T min (Collection <? extends T> c, Comparator comp)
Void reverse (List <T> list )
```

We have already seen synchronizedSet(), synchronizedList() and synchronizedMap() methods.

Collection Framework

Concurrent Collections

The `java.util.concurrent` package includes a number of additions to the Java Collections Framework. These are most easily categorized by the collection interfaces provided:

- `BlockingQueue` defines a first-in-first-out data structure that blocks or times out when you attempt to add to a full queue, or retrieve from an empty queue.
- `ConcurrentMap` is a subinterface of `java.util.Map` that defines useful atomic operations. These operations remove or replace a key-value pair only if the key is present, or add a key-value pair only if the key is absent. Making these operations atomic helps avoid synchronization. The standard general-purpose implementation of `ConcurrentMap` is `ConcurrentHashMap`, which is a concurrent analog of `HashMap`.
- `ConcurrentNavigableMap` is a subinterface of `ConcurrentMap` that supports approximate matches. The standard general-purpose implementation of `ConcurrentNavigableMap` is `ConcurrentSkipListMap`, which is a concurrent analog of `TreeMap`.