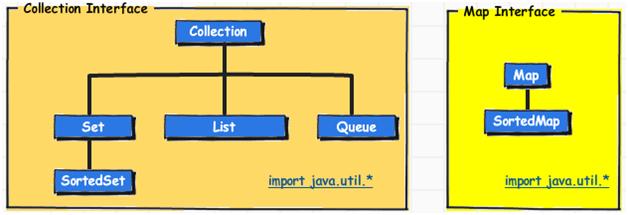
Collections in JAVA

Collection represents the group of objects. Depending on the method of storing and retrieving, collections are basically divided into three parts – [**Set**](http://download.oracle.com/javase/6/docs/api/java/util/Set.html)**,** [**Map**](http://download.oracle.com/javase/6/docs/api/java/util/Map.html) **and** [**List**](http://download.oracle.com/javase/6/docs/api/java/util/List.html). Where Set does **not contain duplicate** values, Map contains **key value** type of data whereas List can have a duplicate values stored in it **sequentially**. This framework is provided in “*java.util*” package. [Collection](http://download.oracle.com/javase/6/docs/api/java/util/Collection.html) is the parent interface of all collections in java.

Following list describes the core collection interfaces.

[](https://jitendrazaa.com/blog/wp-content/uploads/2011/04/Java-Collection-Interfaces.jpg)Java Collection Interfaces

**Collection** “” the **root of the collection** hierarchy. A collection represents a group of objects known as its elements. Some types of collections allow duplicate elements, and others do not. Some are ordered and others are unordered. The Java platform doesn’t provide any direct implementations of this interface but provides implementations of more specific sub interfaces, such as Set and List.

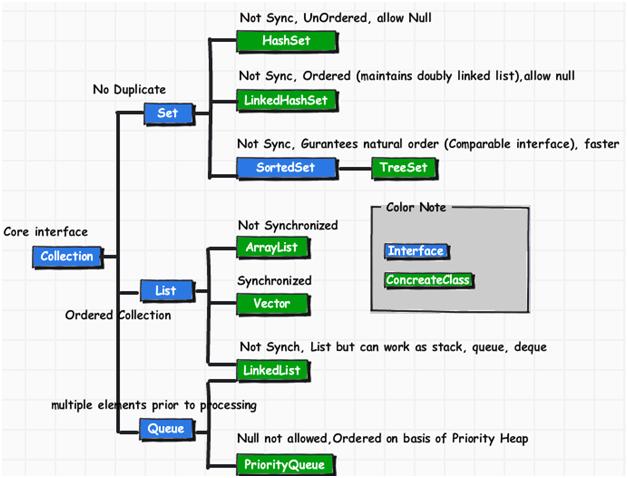
**Set** “” a collection that cannot contain duplicate elements

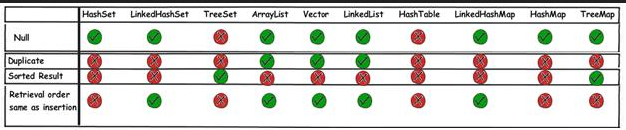
**List** “” an ordered collection (sometimes called a sequence). Lists can contain duplicate elements. The user of a List generally has precise control over where in the list each element is inserted and can access elements by their integer index (position).

**Queue** “” a collection used to hold multiple elements prior to processing. Besides basic Collection operations, a Queue provides additional insertion, extraction, and inspection operations.  
Queues typically, but do not necessarily, order elements in a FIFO (first-in, first-out) manner. Among the exceptions are priority queues, which order elements according to a supplied comparator or the elements’ natural ordering.

**Map** “” an object that maps keys to values. A Map cannot contain duplicate keys; each key can map to at most one value. If you’ve used Hashtable, you’re already familiar with the basics of Map.

**Concrete Classes :**

[](https://jitendrazaa.com/blog/wp-content/uploads/2011/04/Java-Collection-interfaces-and-concrete-classes.jpg)Java Collection interfaces and concrete classes



Every collection type is suitable for a particular scenario. There is no *fastest* or *best* collection.

* If you need fast access to elements using *index*, ArrayList is your answer.
* If you need fast access to elements using a *key*, use HashMap.
* If you need fast add and removal of elements, use LinkedList (but it has a *very* poor index access performance).

and so on.

Null – not allowed in – Tree set and HashTable

Duplicate – allowed in – Lists (Array list, vector and Linked list)

Sorted Result- allowed in – Tree Set and Tree map

Retrieval order same as insertion - allowed in -Lists (Array list, vector and Linked list) and linked hash set and linked hash map.

[**HashSet**](http://download.oracle.com/javase/6/docs/api/java/util/HashSet.html) **:**  
This class implements the Set interface, backed by a hash table (actually a HashMap instance). It makes no guarantees as to the iteration order of the set; in particular, it does not guarantee that the order will remain constant over time. This class *permits the null* element.

[**LinkedHashSet :**](http://download.oracle.com/javase/6/docs/api/java/util/LinkedHashSet.html)  
Hash table and linked list implementation of the Set interface, with predictable iteration order. This implementation differs from HashSet in that it maintains a doubly-linked list running through all of its entries. This linked list defines the iteration ordering, which is the order in which elements were inserted into the set (insertion-order). Note that insertion order is not affected if an element is re-inserted into the set. (An element e is reinserted into a set s if s.add(e) is invoked when s.contains(e) would return true immediately prior to the invocation).

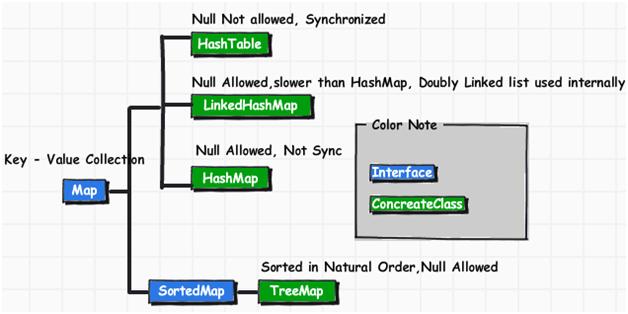
**Interface** [**SortedSet**](http://download.oracle.com/javase/6/docs/api/java/util/SortedSet.html)**:**  
A set that further guarantees that its iterator will traverse the set in ascending element order, sorted according to the natural ordering of its elements (see Comparable), or by a [Comparator](http://download.oracle.com/javase/6/docs/api/java/util/Comparator.html) provided at sorted set creation time. Several additional operations are provided to take advantage of the ordering. (This interface is the set analogue of SortedMap).

[**TreeSet:**](http://download.oracle.com/javase/6/docs/api/java/util/TreeSet.html)  
This class implements the Set interface, backed by a TreeMap instance. This class *guarantees that the sorted set will be in ascending element order,* sorted according to the natural order of the elements (seeComparable), or by the comparator provided at set creation time, depending on which constructor is used. Allows first element as “null”. But NPE will be thrown.

[**ArrayList:**](http://download.oracle.com/javase/6/docs/api/java/util/ArrayList.html)  
Resizable-array implementation of the List interface. Implements all optional list operations, and permits all elements, including null. In addition to implementing the List interface, this class provides methods to manipulate the size of the array that is used internally to store the list.

[**Vector:**](http://download.oracle.com/javase/6/docs/api/java/util/Vector.html)  
This class is roughly equivalent to ArrayList except it is Synchronized.

[**LinkedList:**](http://download.oracle.com/javase/6/docs/api/java/util/LinkedList.html)  
Linked list implementation of the List and Queue interfaces. Implements all optional operations, and permits all elements (including null).

[](https://jitendrazaa.com/blog/wp-content/uploads/2011/04/JAVA-Map-interface-and-concrete-classes.jpg)JAVA Map interface and concrete classes

[**HashMap:**](http://download.oracle.com/javase/6/docs/api/java/util/HashMap.html)  
The HashMap class is roughly equivalent toHashtable, except that it is *unsynchronized and permits nulls*.

[**HashTable:**](http://download.oracle.com/javase/6/docs/api/java/util/Hashtable.html)  
Hash table based implementation of the Map interface. This implementation provides all of the optional map operations, and permits null values and the null key. This class makes *no guarantees as to the order of the map*; in particular, it does not guarantee that the order will remain constant over time.

[**LinkedHashMap:**](http://download.oracle.com/javase/6/docs/api/java/util/LinkedHashMap.html)  
Hash table and linked list implementation of the Map interface, *with predictable iteration order*. This implementation differs from HashMap in that it maintains a doubly-linked list running through all of its entries. This linked list defines the iteration ordering, which is normally the order in which keys were inserted into the map (insertion-order). Note that insertion order is not affected if a key is re-inserted into the map.

[**TreeMap:**](http://download.oracle.com/javase/6/docs/api/java/util/TreeMap.html)  
This class guarantees that the map will be in *ascending key order, sorted according* to the natural order for the key’s class (seeComparable), or by the comparator provided at creation time, depending on which constructor is used.