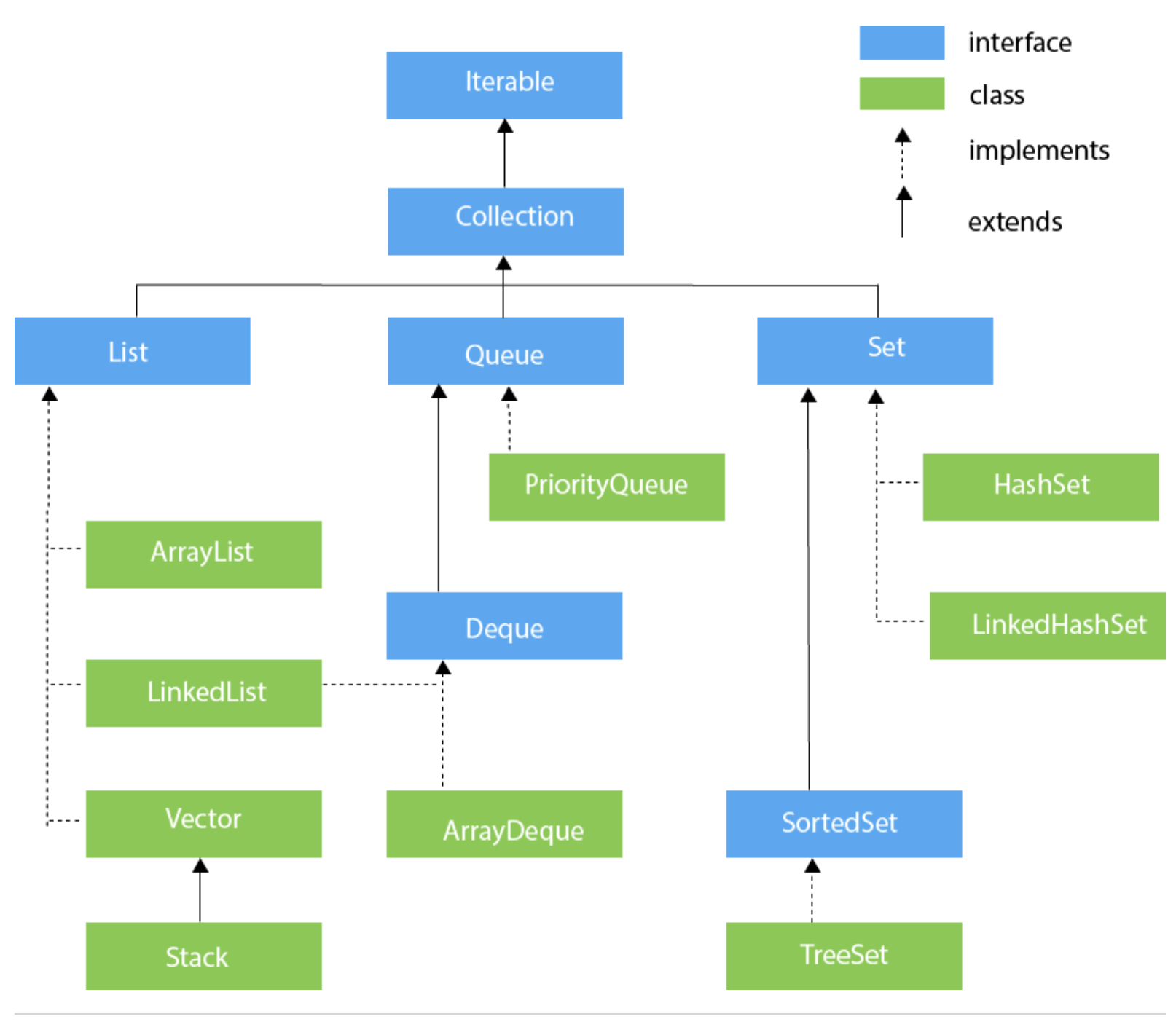
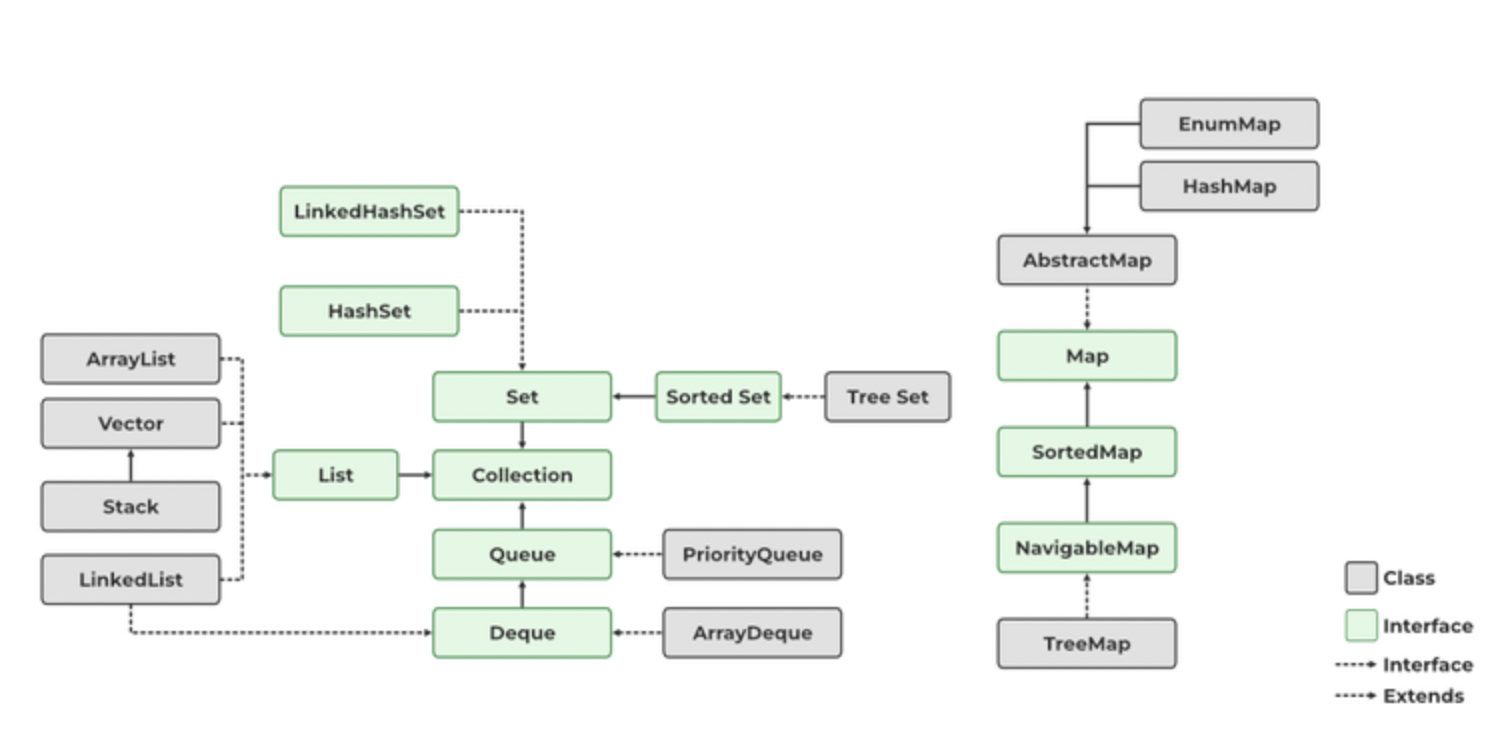
**Generics?**

**Generics** means **parameterized types**. The idea is to allow type (Integer, String, … etc., and user-defined types) to be a parameter to methods, classes, and interfaces. Using Generics, it is possible to create classes that work with different data types. The **Object** is the superclass of all other classes, and Object reference can refer to any object. These features lack type safety. Generics add that type of safety feature.





## **What is Java Collections Framework?**

A *collection* (also known as a *container*) is a single object representing a group of objects. Collections are like containers that group multiple items in a single unit. For example, a jar of chocolates, a list of names, etc.

Collections are used in every programming language and when Java arrived, it also came with few Collection classes – **Vector, Stack, Hashtable, Array**. The Java Collection framework provides many interfaces (**Set, List, Queue, Deque**) and classes (**ArrayList, Vector,** [**LinkedList**](https://www.javatpoint.com/java-linkedlist)**,** [**PriorityQueue**](https://www.javatpoint.com/java-priorityqueue)**, HashSet, LinkedHashSet, TreeSet**).

Java 1.2 provided a Collections Framework that is the architecture to represent and manipulate Collections in java in a standard way. Java Collections Framework consists of the following parts:

### **1. Interfaces**

Java Collections Framework interfaces provide the abstract data type to represent collection.

java.util.Collection is the root interface of Collections Framework. It is on the top of the Collections framework hierarchy. It contains some important methods such as size(), iterator(), add(), remove(), clear() that every Collection class must implement.

Some other important interfaces are java.util.List, java.util.Set, java.util.Queue and java.util.Map. The Map is the only interface that doesn’t inherit from the Collection interface but it’s part of the Collections framework. All the collections framework interfaces are present in the java.util package.

### **2. Implementation Classes**

Java Collections framework provides implementation classes for core collection interfaces. We can use them to create different types of collections in the Java program.

Some important collection classes are ArrayList, LinkedList, HashMap, TreeMap, HashSet, and TreeSet. These classes solve most of our programming needs but if we need some special collection class, we can extend them to create our custom collection class.

Java 1.5 came up with thread-safe collection classes that allowed us to modify Collections while iterating over them. Some of them are CopyOnWriteArrayList, ConcurrentHashMap, CopyOnWriteArraySet. These classes are in java.util.concurrent package.

All the collection classes are present in java.util and java.util.concurrent packages.

### **3. Algorithms**

Algorithms are useful methods to provide some common functionalities such as searching, sorting and shuffling.

## **Java Collections API Interfaces**

Java collection interfaces are the foundation of the Java Collections Framework. Note that all the core collection interfaces are generic; for example public interface Collection<E>. The <E> syntax is for [Generics](https://www.digitalocean.com/community/tutorials/java-generics-example-method-class-interface) and when we declare Collection, we should use it to specify the type of Object it can contain. It helps in reducing run-time errors by type-checking the Objects at compile-time.

To keep the number of core collection interfaces manageable, the Java platform doesn’t provide separate interfaces for each variant of each collection type. If an unsupported operation is invoked, a collection implementation throws an UnsupportedOperationException.

### **1. Collection interface**

This is the root of the collection hierarchy. A collection represents a group of objects known as its elements. The Java platform doesn’t provide any direct implementations of this interface.

The interface has methods to tell you how many elements are in the collection (size, isEmpty), to check whether a given object is in the collection (contains), to add and remove an element from the collection (add, remove), and to provide an iterator over the collection (iterator).

Collection interface also provides bulk operations methods that work on the entire collection – containsAll, addAll, removeAll, retainAll, clear.

The toArray methods are provided as a bridge between collections and older APIs that expect arrays on input.

### **2. Iterator Interface**

Iterator interface provides methods to iterate over the elements of the Collection. We can get the instance of an iterator using the iterator() method. Iterator takes the place of Enumeration in the Java Collections Framework. Iterators allow the caller to remove elements from the underlying collection during the iteration. Iterators in collection classes implement the Iterator Design Pattern.

### **3. Set Interface**

Set is a collection that cannot contain duplicate elements. This interface models the mathematical set abstraction and is used to represent sets, such as the deck of cards.

The Java platform contains three general-purpose Set implementations: [**HashSet**](https://www.digitalocean.com/community/tutorials/java-hashset)**, TreeSet, and LinkedHashSet**. Set interface doesn’t allow random-access to an element in the Collection. You can use an iterator or foreach loop to traverse the elements of a Set.

### **4. List Interface**

[List](https://www.digitalocean.com/community/tutorials/java-list) is an ordered collection and can contain duplicate elements. You can access any element from its index. List is more like an array with dynamic length. List is one of the most used Collection types. **ArrayList and** [**LinkedLis**t](https://www.digitalocean.com/community/tutorials/java-linkedlist-linkedlist-java) are implementation classes of List interface.

List interface provides useful methods to add an element at a specific index, remove/replace element based on the index and to get a sub-list using the index.

Collections class provides some useful algorithms for List – sort, shuffle, reverse, binarySearch etc.

### **5. Queue Interface**

[Queue](https://www.digitalocean.com/community/tutorials/java-queue) is 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. Whatever the ordering used, the head of the queue is the element that would be removed by a call to remove or poll. In a FIFO queue, all new elements are inserted at the tail of the queue.

### **6. Dequeue Interface**

A linear collection that supports element insertion and removal at both ends. The name deque is short for “double-ended queue” and is usually pronounced “deck”. Most Deque implementations place no fixed limits on the number of elements they may contain, but this interface supports capacity-restricted deques as well as those with no fixed size limit.

This interface defines methods to access the elements at both ends of the deque. Methods are provided to insert, remove, and examine the element.

### **7. Map Interface**

[Java Map](https://www.digitalocean.com/community/tutorials/java-map) is an object that maps keys to values. A map cannot contain duplicate keys: Each key can map to at most one value.

The Java platform contains three general-purpose Map implementations: HashMap, TreeMap, and LinkedHashMap.

The basic operations of Map are put, get, containsKey, containsValue, size, and isEmpty.

### **8. ListIterator Interface**

An iterator for lists that allows the programmer to traverse the list in either direction, modify the list during iteration, and obtain the iterator’s current position in the list.

[Java ListIterator](https://www.digitalocean.com/community/tutorials/java-listiterator) has no current element; its cursor position always lies between the element that would be returned by a call to previous() and the element that would be returned by a call to next().

### **9. SortedSet Interface**

SortedSet is a Set that maintains its elements in ascending order. Several additional operations are provided to take advantage of the ordering. Sorted sets are used for naturally ordered sets, such as word lists and membership rolls.

### **10. SortedMap Interface**

A map that maintains its mappings in ascending key order. This is the Map analog of SortedSet. Sorted maps are used for naturally ordered collections of key/value pairs, such as dictionaries and telephone directories.

## **Java Collections Classes**

The Java Collections framework comes with many implementation classes for the interfaces. Most common implementations are **ArrayList, HashMap and HashSet**. Java 1.5 included Concurrent implementations; for example ConcurrentHashMap and CopyOnWriteArrayList. Usually Collection classes are not thread-safe and their iterator is fail-fast. In this section, we will learn about commonly used collection classes.

### **1. HashSet Class**

[Java HashSet](https://www.digitalocean.com/community/tutorials/java-hashset) is the basic implementation of the Set interface that is backed by a [HashMap](https://www.digitalocean.com/community/tutorials/java-hashmap). It makes no guarantees for iteration order of the set and permits the null element.

This class offers constant time performance for basic operations (add, remove, contains and size), assuming the hash function disperses the elements properly among the buckets. We can set the initial capacity and load factor for this collection. The load factor is a measure of how full the hash map is allowed to get before its capacity is automatically increased.

### **2. TreeSet Class**

A NavigableSet implementation based on a TreeMap. The elements are ordered using their natural ordering, or by a Comparator provided at set creation time, depending on which constructor is used.

Refer: Java Comparable Comparator

This implementation provides guaranteed log(n) time cost for the basic operations (add, remove, and contains).

Note that the ordering maintained by a set (whether or not an explicit comparator is provided) must be consistent with equals if it is to correctly implement the Set interface. (See Comparable or Comparator for a precise definition consistent with equals.) This is so because the Set interface is defined in terms of the equals operation, but a TreeSet instance performs all element comparisons using its compareTo (or compare) method, so two elements that are deemed equal by this method are, from the standpoint of the set, equal.

### **3. ArrayList Class**

[Java ArrayList](https://www.digitalocean.com/community/tutorials/java-arraylist) is the 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. (This class is roughly equivalent to Vector, except that it is unsynchronized.)

The size, isEmpty, get, set, iterator, and list iterator operations run in constant time. The add operation runs in amortized constant time, that is, adding n elements requires O(n) time. All of the other operations run in linear time (roughly speaking). The constant factor is low compared to that for the LinkedList implementation.

### **4. LinkedList Class**

Doubly-linked list implementation of the List and Deque interfaces. Implements all optional list operations, and permits all elements (including null).

All of the operations perform as expected for a doubly-linked list. Operations that index into the list will traverse the list from the start or the end, whichever is closer to the specified index.

### **5. HashMap Class**

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. HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits null. This class makes no guarantees for the order of the map.

This implementation provides constant-time performance for the basic operations (get and put). It provides constructors to set initial capacity and load factor for the collection.

### **6. TreeMap Class**

A Red-Black tree based NavigableMap implementation. The map is sorted according to the natural ordering of its keys, or by a Comparator provided at map creation time, depending on which constructor is used.

This implementation provides guaranteed log(n) time cost for the containsKey, get, put, and remove operations. Algorithms are adaptations of those in Cormen, Leiserson, and Rivest’s Introduction to Algorithms.

Note that the ordering maintained by a TreeMap, like any sorted map, and whether or not an explicit comparator is provided, must be consistent with equals if this sorted map is to correctly implement the Map interface. (See Comparable or Comparator for a precise definition consistent with equals.) This is so because the Map interface is defined in terms of the equals operation, but a sorted map performs all key comparisons using its compareTo (or compare) method, so two keys that are deemed equal by this method are, from the standpoint of the sorted map, equal. The behavior of a sorted map is well-defined even if its ordering is inconsistent with equals; it just fails to obey the general contract of the Map interface.

### **7. PriorityQueue Class**

Queue processes its elements in FIFO order but sometimes we want elements to be processed based on their priority. We can use PriorityQueue in this case and we need to provide a Comparator implementation while instantiating the PriorityQueue. PriorityQueue doesn’t allow null values and it’s unbounded. For more details about this, please head over to [Java Priority Queue](https://www.digitalocean.com/community/tutorials/java-priority-queue-priorityqueue-example) where you can check its usage with a sample program.