**What is a Collections Framework?**

A collections framework is a unified architecture for representing and manipulating collections.

All collections frameworks contain the following:

* **Interfaces:** These are abstract data types that represent collections. Interfaces allow collections to be manipulated independently of the details of their representation. In object-oriented languages, interfaces generally form a hierarchy.
* **Implementations:** These are the concrete implementations of the collection interfaces. In essence, they are reusable data structures.
* **Algorithms:** These are the methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces. The algorithms are said to be polymorphic: that is, the same method can be used on many different implementations of the appropriate collection interface. In essence, algorithms are reusable functionality.

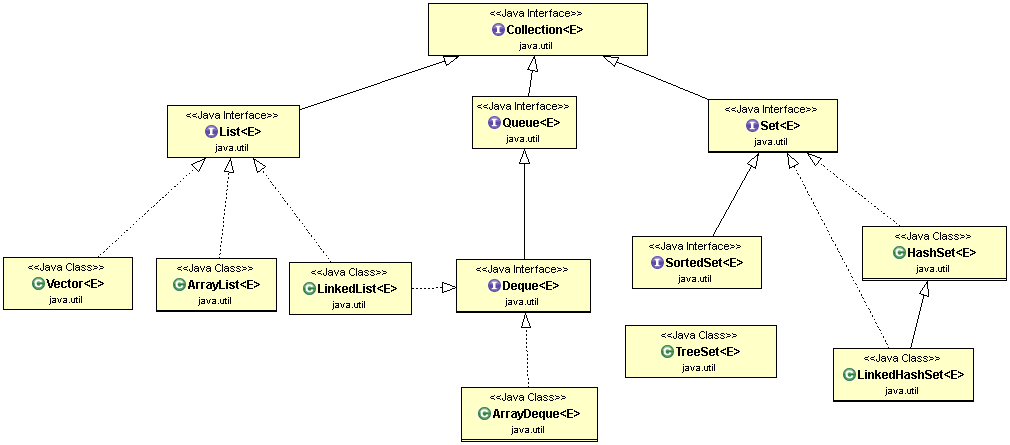
**Benefits of the Java Collections Framework**

The Java Collections Framework provides the following benefits:

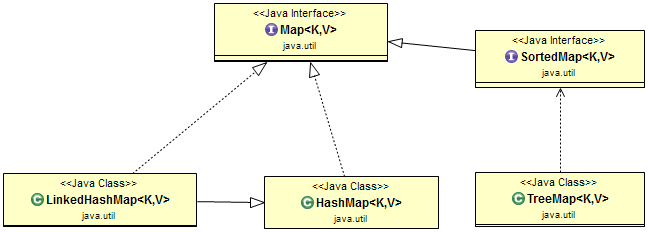
1. **Reduces programming effort** by providing data structures and algorithms so you don't have to write them yourself.
2. **Increases performance** by providing high-performance implementations of data structures and algorithms. Because the various implementations of each interface are interchangeable, programs can be tuned by switching implementations.
3. **Provides interoperability** between unrelated APIs by establishing a common language to pass collections back and forth.
4. **Reduces the effort required to learn APIs** by requiring you to learn multiple ad hoc collection APIs.
5. **Reduces the effort required to design and implement APIs** by not requiring you to produce ad hoc collections APIs.
6. **Fosters software reuse** by providing a standard interface for collections and algorithms with which to manipulate them.

**Hierarchy of Collection Framework**

Let us see the hierarchy of the Collection framework. The *java.util* package contains all the classes and interfaces for the Collection framework.  
  
The collection interfaces are divided into two groups:  
  
The most basic interface, **java.util.Collection**.

****

 The other collection interfaces are based on *java.util.Map* and are not true collections:

****

**Collection Interfaces**

The collection interfaces are divided into two groups. The most basic interface, *java.util.Collection*, has the following descendants:

* java.util.Collection
* java.util.List
* java.util.Set
* java.util.SortedSet
* java.util.NavigableSet
* java.util.Queue
* java.util.concurrent.BlockingQueue
* java.util.concurrent.TransferQueue
* java.util.Deque
* java.util.concurrent.BlockingDeque

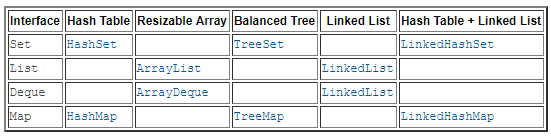
The other collection interfaces are based on *java.util.Map* and are not true collections. However, these interfaces contain collection-view operations, which enable them to be manipulated as collections.

Interfaces related to Map:

* java.util.Map
* java.util.SortedMap
* java.util.NavigableMap
* java.util.concurrent.ConcurrentMap
* java.util.concurrent.ConcurrentNavigableMap

**Collection Implementations**

The general-purpose collection interface implementations are summarized in the following table:

****

The *AbstractCollection, AbstractSet, AbstractList, AbstractSequentialList and AbstractMap*classes provide basic implementations of the core collection interfaces, to minimize the effort required to implement them.

The API documentation for these classes describes precisely how each method is implemented so the implementer knows which methods must be overridden, given the performance of the basic operations of a specific implementation.

**Commonly used implementations**

The Java Collections Framework provides several general-purpose implementations of the core interfaces:

* For the **Set** interface, HashSet is the most commonly used implementation.
* For the **List** interface, ArrayList is the most commonly used implementation.
* For the **Map** interface, HashMap is the most commonly used implementation.
* For the Queue interface, LinkedList is the most commonly used implementation.
* For the Deque interface, ArrayDeque is the most commonly used implementation.

**Interfaces**

* **Collections Framework - The Collection Interface**
* **Collections Framework - The Set Interface**
* **Collections Framework - The SortedSet Interface**
* **Collections Framework - The List Interface**
* **Collections Framework - The Queue Interface**
* **Collections Framework - The Deque Interface**
* **Collections Framework - The Map Interface**
* **Collections Framework - The SortedMap Interface**

**List Implementations**

**General-Purpose List Implementations:**

* **Collections Framework - ArrayList Class**
* **Collections Framework - LinkedList Class**

**Special-Purpose List Implementations:**

* **Collections Framework - CopyOnWriteArrayList**

**Set Implementations**

**General-Purpose Set Implementations:**

* **Collections Framework - HashSet Class**
* **Collections Framework - LinkedHashSet Class**
* **Collections Framework - TreeSet Class**

**Special-Purpose Set Implementations:**

* **Collections Framework - CopyOnWriteArraySet**
* **Collections Framework - EnumSet**

**Map Implementations**

**General-purpose Map Implementations:**

* **Collections Framework - HashMap Class**
* **Collections Framework - LinkedHashMap Class**
* **Collections Framework - TreeMap class**

**Special-Purpose Map Implementations:**

* **Collections Framework - EnumMap**
* **Collections Framework - WeakHashMap**
* **Collections Framework - IdentityHashMap**