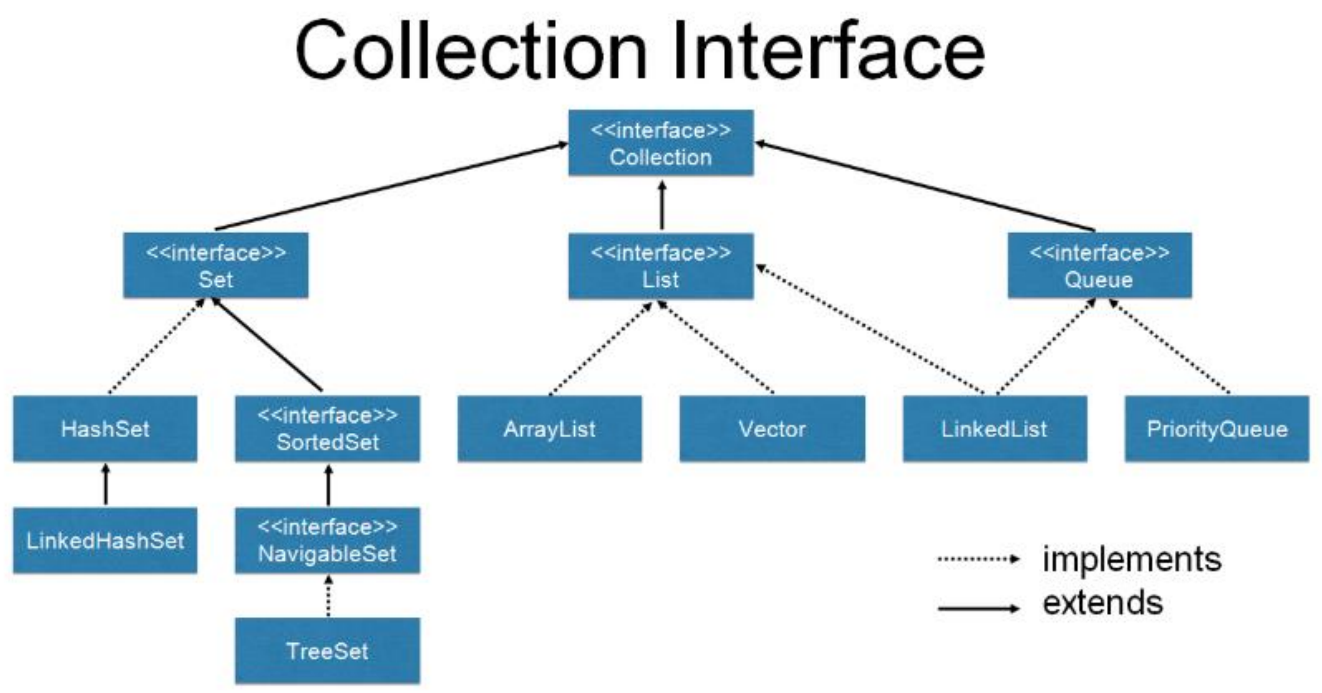


HashSet, LinkedHasSet, and TreeSet

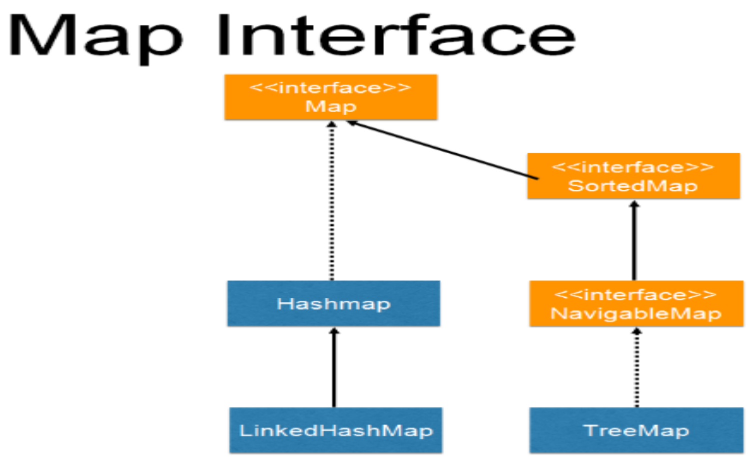
1. **HashSet** is the default implementation used in most cases.
2. **LinkedHashSet** is like a combination of HashSet and List in that it does not allow duplicate entries as with Sets, but traverses its elements in the order they were inserted, like a List would do.
3. **TreeSet** will constantly keep all its elements in some sorted order. Keep in mind, however, that there is no such thing as a *free lunch* and that every added feature comes at a certain cost.

List Interface

ArrayList is the default implementation for List, located to the middle of the collection hierarchy in Figure. Like any List implementation, it does allow duplicate elements and iteration in the order of insertion. As it is based on arrays, it is very fast to **iterate and read** from, but very slow to **add or remove** an element at random positions, as it has to rebuild the underlying array structure.
In contrast,
LinkedList makes it easy to add or remove elements at any position in the list while being slower to read from at random positions.

On Vectors

As a side note, we shortly consider Vector, a class that has been around since JDK 1, even before the Collections Framework which was added with Java 2. ***Long story short****, its performance is suboptimal, so no new code should ever have to use it. An ArrayList or LinkedList simply does a better job.*



Map Interface oddly enough has no relation to the Collection interface.   A Collection operates on **one** entity, while a Map operates on **two**: a unique **key**(a vehicle identification number) *and* a **value** object related to the key(a car).

To retrieve an object from a Map, you would normally use its key. Map is the root of quite a number of interfaces and classes, as depicted on Figure.

Hashtable, Hashmap, and LinkedHashMap

The Hashtable class was the first Collection in Java 1 that was based on the hash-table data structure. Unfortunately, like Vector, the class is deprecated because of its suboptimal performance. We can forget about it and use the other Map implementations instead. **HashMap** is the default implementation that you will find yourself using in most cases.

A **Map** usually doesn't make any guarantee as to how it internally stores elements. An exception to this rule, however, is **LinkedHashMap**, which allows us to iterate the map in the order of insertion.