**What is a Framework?**

A framework is a set of classes and interfaces which provide a ready-made architecture. In order to implement a new feature or a class, there is no need to define a framework. However, an optimal object-oriented design always includes a framework with a collection of classes such that all the classes perform the same kind of task.

**Collection:** Collection is an interface present in java.util.package. It is used to represent a group of individual objects as a single unit. It is similar to the container in the C++ language. The collection is ***considered as the root interface of the collection framework***. It provides several classes and interfaces to represent a group of individual objects as a single unit.

Any group of individual objects which are represented as a single unit is known as the collection of the objects. In Java, a separate framework named the *“Collection Framework”* has been defined in JDK 1.2 which holds all the collection classes and interface in it.

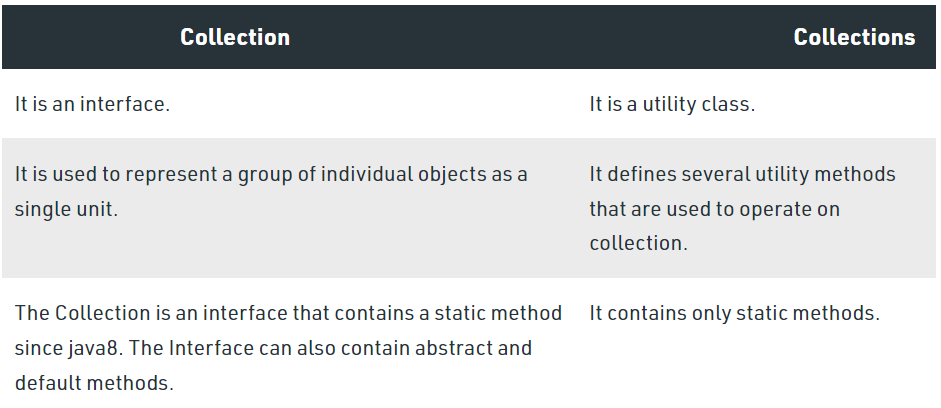
The Collection interface (**java.util.Collection**) and Map interface (**java.util.Map**) are the two main “root” interfaces of Java collection classes.

Collection is a generic interface that has this declaration:

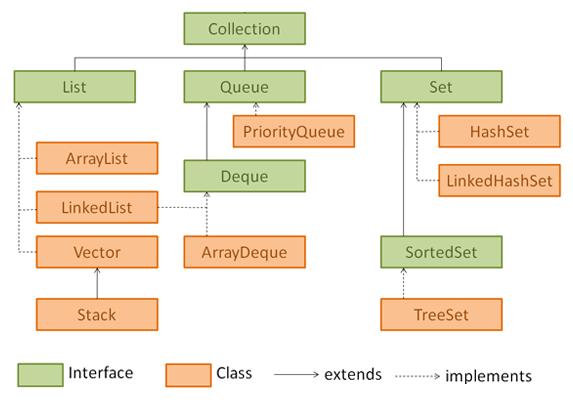
**interface Collection<E>**

Here, E specifies the type of objects that the collection will hold. Collection extends the Iterable interface. This means that all collections can be cycled through by use of the foreach style for loop. (Recall that only classes that implement Iterable can be cycled through by the for.)

**Collections:** Collections is a utility class present in java.util.package. It defines several utility methods like sorting and searching which is used to operate on collection. It has all static methods. These methods provide much-needed convenience to developers, allowing them to effectively work with Collection Framework.



Pictorial representation of Collection



Collection Interface defines most common methods which are applicable for any collection object. There is not Concreate that implements collection interface directly.

1. **List(I):**

It is the child interface of collection. If we want to represent a group of individual object as a single entity where duplicates are allowed and insertion order must be preserved then we should for List.

**Implementation Class of List Interface**

1. Array List (**v1.2**)
2. Linked List (**v1.2**)
3. Vectors 🡨 Stack (Lagacy Classes) **v1.0**

**Note: In 1.2 version vector and Stack classes are re-engineered to implement List interface.**

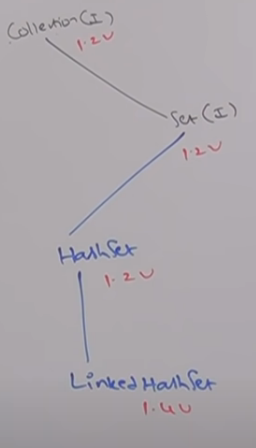
1. **Set(I):**

It is the child interface of collection. If we want to represent a group of individual Objects as a single entity where duplicates are not allowed and insertion order not required then we should go for set.

Implementation class of Set Interface

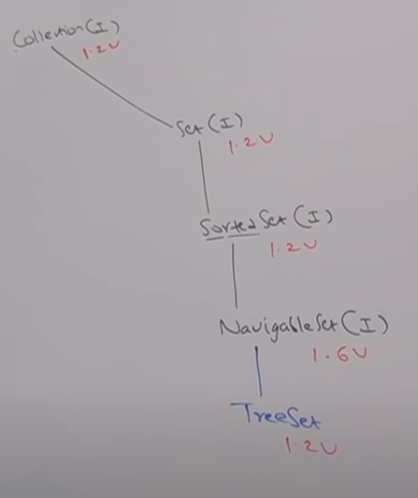
HashSet **v1.2**

Linked HashSet **v1.4**



**Sorted set(I) v1.2:** It is the child interface of set If we want to represent a group of individual Objects as a single entity where duplicates are not allowed and all objects should be inserted according to some sorting order then we should go for sorted set.

**Navigable Set(I) v1.6:**  It is the child interface of sorted set; it contains several methods for navigation purpose. Tree set is implementation class for navigable Set interface(I).



1. **Queue(I):**

It is the child interface of collection. If we want to represent a group of individual Objects as a single entity prior to processing then we should go Queue. Usually, Queue fallows FIFO but based on are requirement we can implement or own priority order also.

Example: Before sending a mail all mail IDs we have to store in some data structure. In which order we add mail IDs in the same order only the mail should be delivered for this requirements Queue is the best choice.

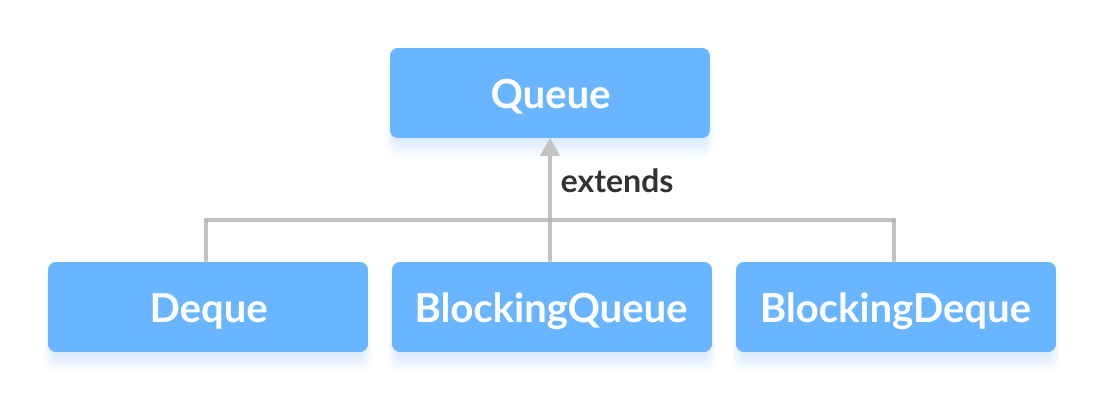
Since the Queue is an interface, we cannot provide the direct implementation of it.

In order to use the functionalities of Queue, we need to use classes that implement it:

* ArrayDeque
* LinkedList
* PriorityQueue

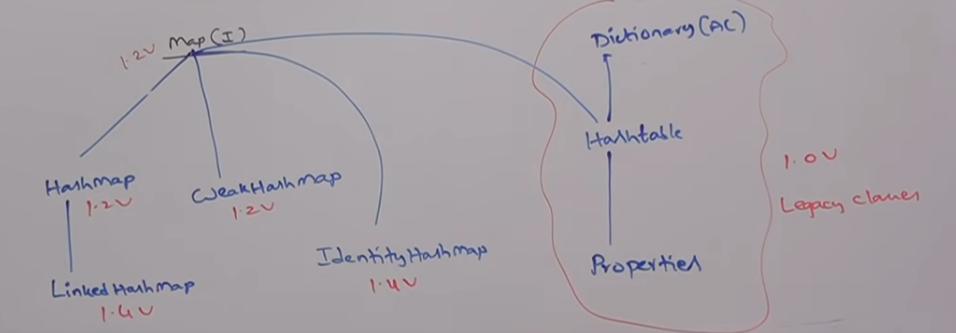
Interfaces that extend Queue: The Queue interface is also extended by various sub interfaces:

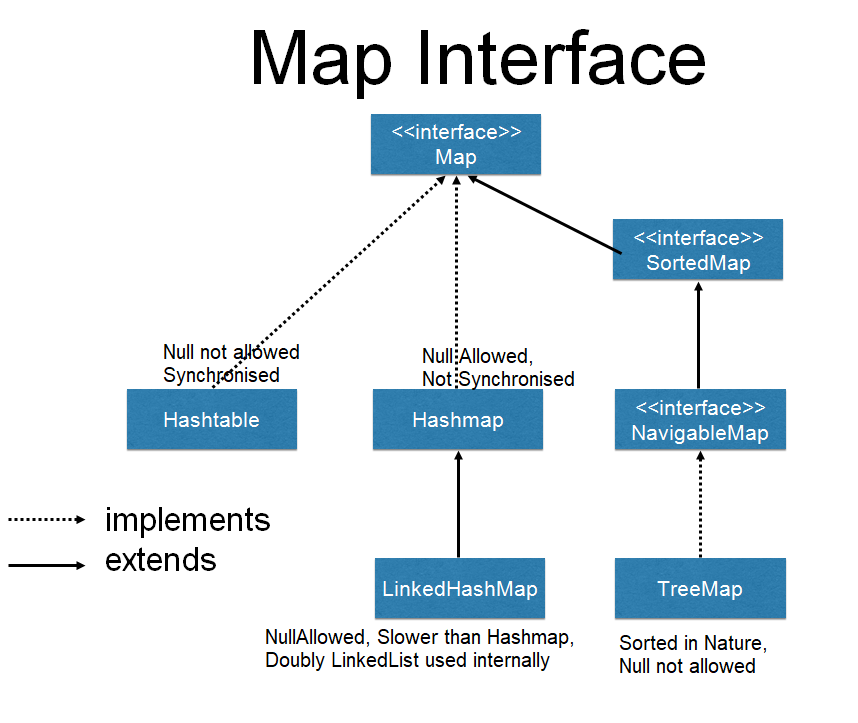
* Deque
* BlockingQueues
* BlockingDeque



1. **Map(I):**

Is **not** the child interface of collection. If we want to represent a group of objects as key-value pairs then we need to go for Map. Both key and value are objects only. Duplicate keys are not allowed but values can be duplicated.





Collection declares the core methods that all collections will have.

