


```
| elem |           | _____ |           | _____ |
| _____ |
| _____ |
| _____ |
| Last
|-----|
|_elem_|
```

Queue

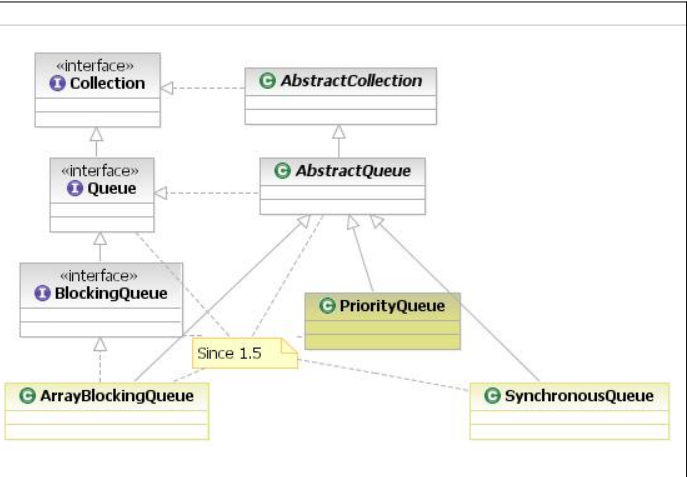
The Queue interface adds the following operations to the Collection interface:

E element()	Retrieves, but does not remove, the head of this queue. This method differs from the peek method only in that it throws an exception if this queue is empty
boolean offer(E o)	Inserts the specified element into this queue, if possible.
E peek()	Retrieves, but does not remove, the head of this queue, returning null if this queue is empty
E poll()	Retrieves and removes the head of this queue, or null if this queue is empty
E remove()	Retrieves and removes the head of this queue. This method differs from the poll method in that it throws an exception if this queue is empty.

java.util.PriorityQueue<E>
orders elements according to an order/priority specified at construction time, null element is not allowed.

java.util.concurrent.ArrayBlockingQueue<E>
orders elements FIFO; synchronized, thread safe.

java.util.concurrent.SynchronousQueue<E>
each put must wait for a take, and vice versa, does not have any internal capacity, not even a capacity of one, an element is only present when you try to take it; you cannot add an element (using any method) unless another thread is trying to remove it



Map Classes

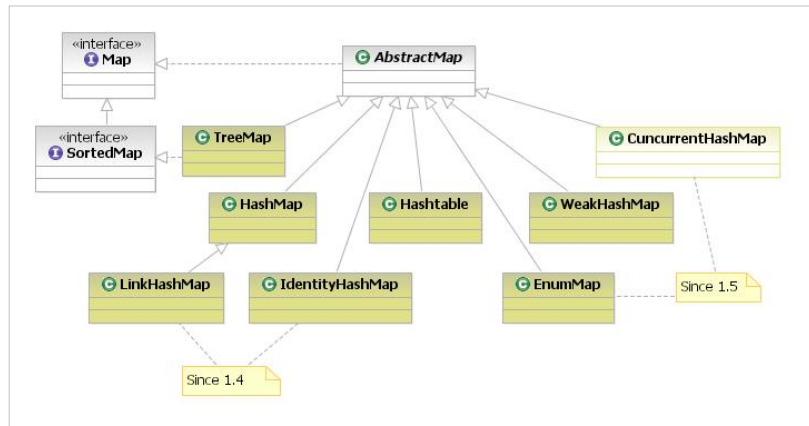
The Map interface has the following implementations:

`java.util.TreeMap<E>`

guarantees that the map will be in ascending key order, sorted according to the natural order for the key's class, not-synchronized.

`java.util.HashMap<E>`

is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls



`java.util.concurrent.ConcurrentHashMap`

same Hashtable, plus retrieval operations (including get) generally do not block, so may overlap with update operations (including put and remove).

`java.util.Hashtable<E>`

Synchronized, null can not be used as key

`java.util.WeakHashMap<E>`

entry in a WeakHashMap will automatically be removed when its key is no longer in ordinary use. Nonsynchronized.

`java.util.LinkHashMap<E>`

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.

`java.util.IdentityHashMap`

This class implements the Map interface with a hash table, using reference-equality in place of object-equality when comparing keys (and values). In other words, in an IdentityHashMap, two keys k1 and k2 are considered equal if and only if (k1==k2). (In normal Map implementations (like HashMap) two keys k1 and k2 are considered equal if and only if (k1==null ? k2==null : k1.equals(k2)).) Not-synchronized.

`java.util.EnumMap`

All of the keys in an enum map must come from a single enum type that is specified, explicitly or implicitly, when the map is created. Enum maps are represented internally as arrays. This representation is extremely compact and efficient. Not-synchronized.

Thread Safe Collections

It is also called Concurrent Collections. Most of the popular collection classes have implementations for both single thread and multiple thread environments. The non-synchronized implementations are always faster. You can use the non-synchronized implementations in multiple thread environments, when you make sure that only one thread updating the collection an any given time.

A new Java JDK package was introduced at Java 1.5, that is `java.util.concurrent`. This package supplies a few Collection implementations designed for use in multithreaded environments.

The following table list all the synchronized collection classes: