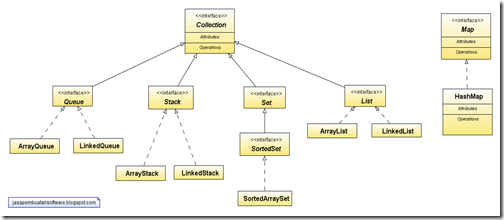
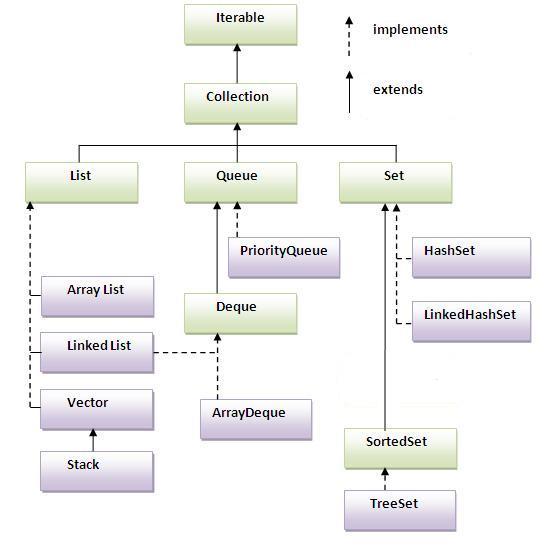
|  | | **Implementations** | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **Hash Table** | **Resizable Array** | **Balanced Tree** | **Linked List** | **Hash Table + Linked List** |
| **Interfaces** | **Set** | [HashSet](http://docs.oracle.com/javase/6/docs/api/java/util/HashSet.html) |  | [TreeSet](http://docs.oracle.com/javase/6/docs/api/java/util/TreeSet.html) |  | [LinkedHashSet](http://docs.oracle.com/javase/6/docs/api/java/util/LinkedHashSet.html) |
| **List** |  | [ArrayList](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html) |  | [LinkedList](http://docs.oracle.com/javase/6/docs/api/java/util/LinkedList.html) |  |
| **Deque** |  | [ArrayDeque](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayDeque.html) |  | [LinkedList](http://docs.oracle.com/javase/6/docs/api/java/util/LinkedList.html) |  |
| **Map** | [HashMap](http://docs.oracle.com/javase/6/docs/api/java/util/HashMap.html) |  | [TreeMap](http://docs.oracle.com/javase/6/docs/api/java/util/TreeMap.html) |  | [LinkedHashMap](http://docs.oracle.com/javase/6/docs/api/java/util/LinkedHashMap.html) |

.





**Public class Stack<E> extends Vector<E>**

* The Stack class represents a last-in-first-out (LIFO) stack of objects.
* Algorithm Used : ?
* It extends class Vector with five operations that allow a vector to be treated as a stack.
* These methods internally makes use of Vector methods such as addElement(), elementAt() and removeElementAt()
* Following are the 5 methods which are added by the Stack

1. public E push(E item)

Pushes an item onto the top of this stack. This has exactly the same effect as “synchronized void addElement (item)” of Vector

1. public synchronized E peek()

Looks at the object at the top of the stack without removing it from the stack.

1. public synchronized E pop()

Removes the object at the top of this stack and returns that object as the value of this function.

1. public synchronized int search(Object o)

This method searches the stack for an item and discover how far it is from the top. It returns the 1-based position where an object is on this stack. If the object <tt>o</tt> occurs as an item in this stack, this method returns the distance from the top of the stack of the occurrence nearest the top of the stack; the topmost item on the stack is considered to be at distance <tt>1</tt>. The <tt>equals</tt> method is used to compare <tt>o</tt> to the items in this stack.

1. public boolean empty()

Tests if this stack is empty.

The sorting algorithm is a modified mergesort (in which the merge is

\* omitted if the highest element in the low sublist is less than the

\* lowest element in the high sublist). This algorithm offers guaranteed

\* n\*log(n) performance.

/\*\*

\* Searches the specified array of longs for the specified value using the

\* binary search algorithm. The array must be sorted (as

\* by the {@link #sort(long[])} method) prior to making this call. If it

\* is not sorted, the results are undefined. If the array contains

\* multiple elements with the specified value, there is no guarantee which

\* one will be found.

\*

\* @param a the array to be searched

\* @param key the value to be searched for

\* @return index of the search key, if it is contained in the array;

\* otherwise, <tt>(-(<i>insertion point</i>) - 1)</tt>. The

\* <i>insertion point</i> is defined as the point at which the

\* key would be inserted into the array: the index of the first

\* element greater than the key, or <tt>a.length</tt> if all

\* elements in the array are less than the specified key. Note

\* that this guarantees that the return value will be &gt;= 0 if

\* and only if the key is found.

\*/

public static int binarySearch(long[] a, long key) {

return binarySearch0(a, 0, a.length, key);

}

/\*\*

\* Sorts the specified array of longs into ascending numerical order.

\* The sorting algorithm is a tuned quicksort, adapted from Jon

\* L. Bentley and M. Douglas McIlroy's "Engineering a Sort Function",

\* Software-Practice and Experience, Vol. 23(11) P. 1249-1265 (November

\* 1993). This algorithm offers n\*log(n) performance on many data sets

\* that cause other quicksorts to degrade to quadratic performance.

\*

\* @param a the array to be sorted

\*/

public static void sort(long[] a) {

sort1(a, 0, a.length);

}