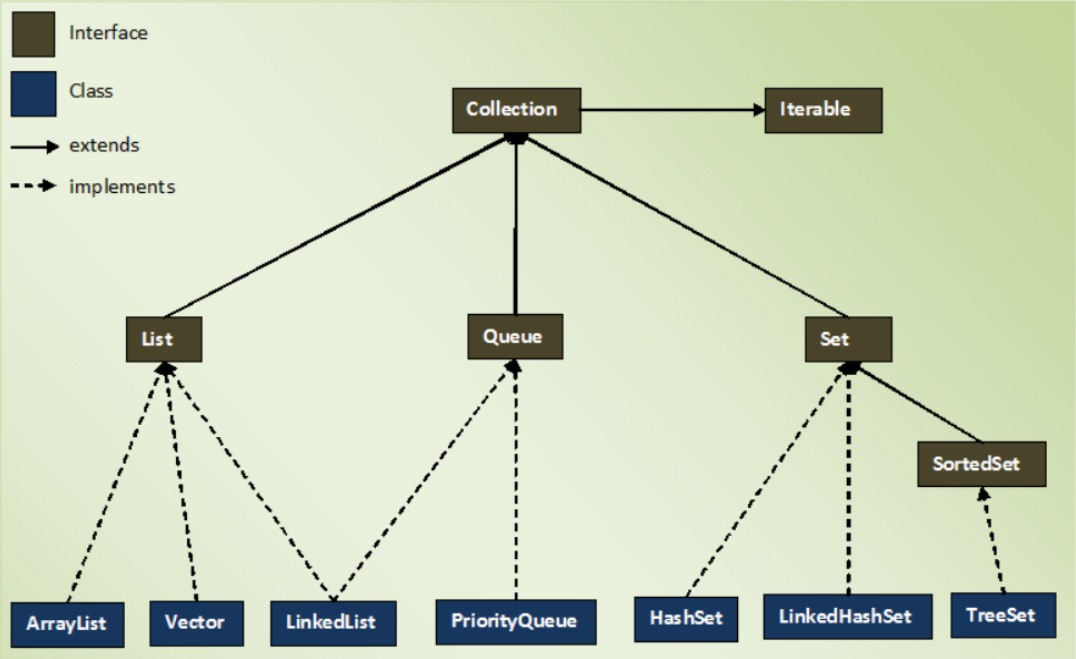
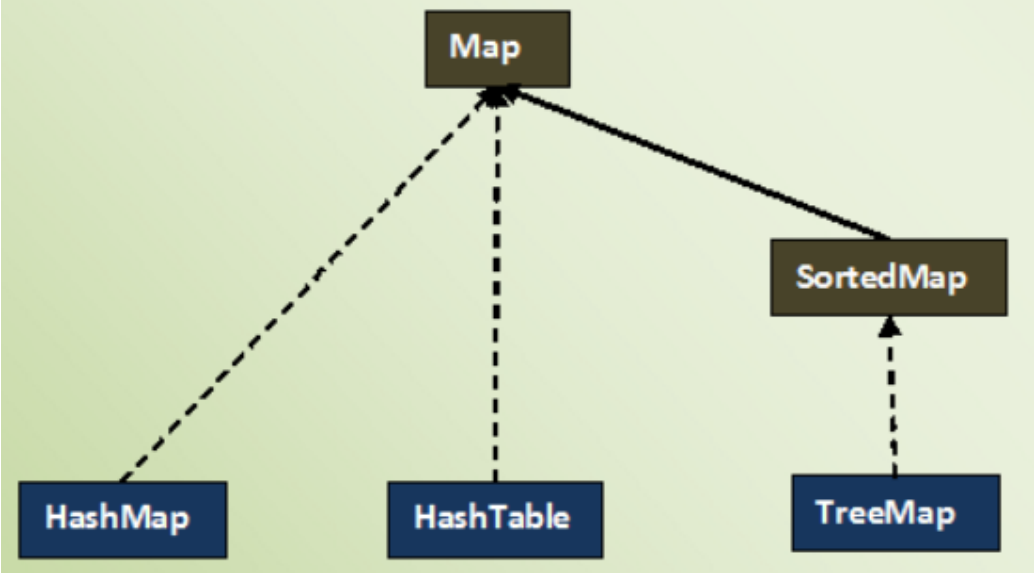
Collections are nothing but a group of objects stored in a well defined manner. Earlier, Arrays are used to represent these groups of objects. But, arrays are not resizable. size of the arrays are fixed. Size of the arrays can not be changed once they are defined. This causes lots of problems while handling a group of objects. To overcome this drawback of arrays, Collection framework or simply collections are introduced in java from JDK 1.2.

Collection Framework in java is a centralized and unified theme to store and manipulate the group of objects. Java Collection Framework provides some predefined classes and interfaces to handle the group of objects. Using a collection framework, you can store the objects as a **list** or as a **set** or as a **queue** or as a **map** and perform operations like adding an object or removing an object or sorting the objects without much hard work.

All classes and interfaces related to Collection Framework are placed in the java.util package. java.util.Collection interface is at the top of class hierarchy of Collection Framework



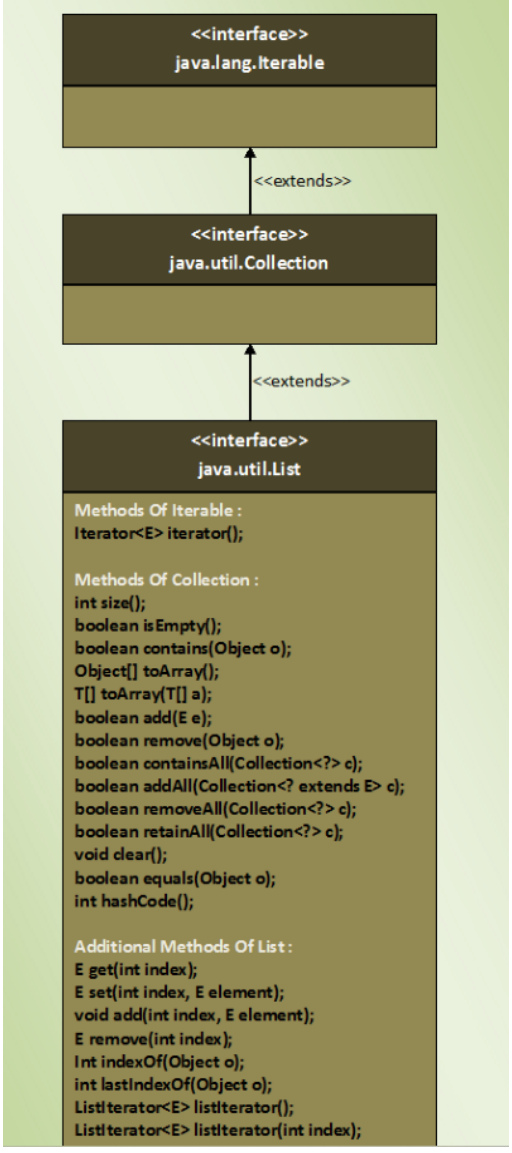


List Interface

List Interface represents an ordered or sequential collection of objects. This interface has some methods which can be used to store and manipulate the ordered collection of objects. The classes which implement the List interface are called as Lists. ArrayList, Vector and LinkedList are some examples of lists. You have the control over where to insert an element and from where to remove an element in the list.

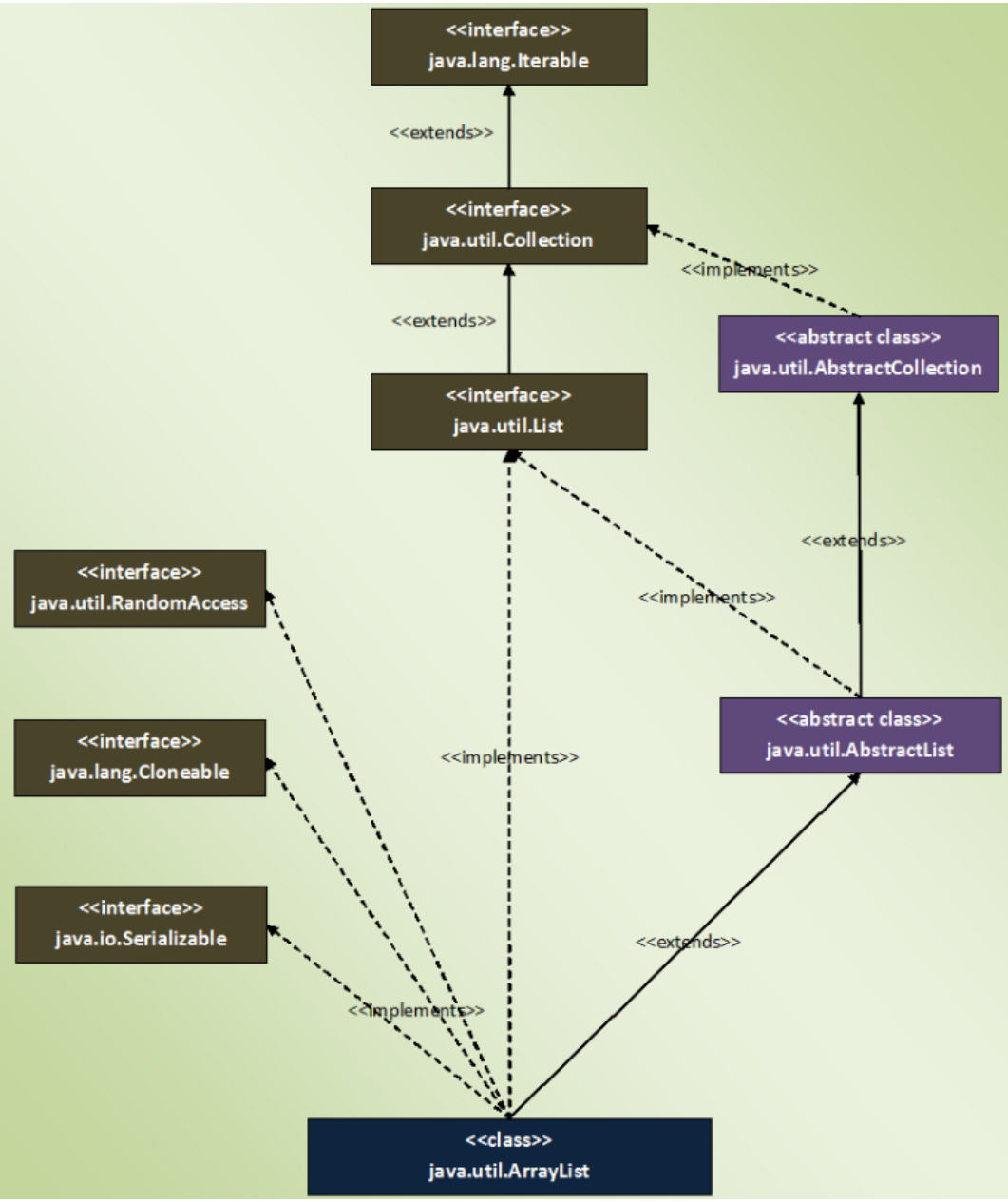
Elements of the lists are ordered using a Zero based index.You can access the elements of lists using an integer index.Elements can be inserted at a specific position using integer index. Any pre-existing elements at or beyond that position are shifted right.Elements can be removed from a specific position. The elements beyond that position are shifted left.A list may contain duplicate elements.A list may contain multiple null elements.

List interface extends Collection interface. So, All 15 methods of Collection interface are inherited to List interface. Along with these methods, another 9 methods are included in the List interface to support the properties of lists.



ArrayList Class

In java, normal arrays are of fixed length. You can not change the size of arrays once they are defined. That means, you must know in advance how large an array you want. But sometimes, you may not know how large an array you want. To overcome this situation, ArrayList is introduced in Collection framework. ArrayList, in simple terms, can be defined as a resizable array. ArrayList is same like normal array but it can grow and shrink dynamically to hold any number of elements. ArrayList is a sequential collection of objects which increases or decreases in size as we add or delete the elements.In ArrayList, elements are positioned according to Zero-based index. That means, elements are inserted from index 0. Default initial capacity of an ArrayList is 10. This capacity increases automatically as we add more elements to arraylist. You can also specify initial capacity of an ArrayList while creating it.ArrayList class implements List interface and extends AbstractList. It also implements 3 marker interfaces – RandomAccess, Cloneable and Serializable.

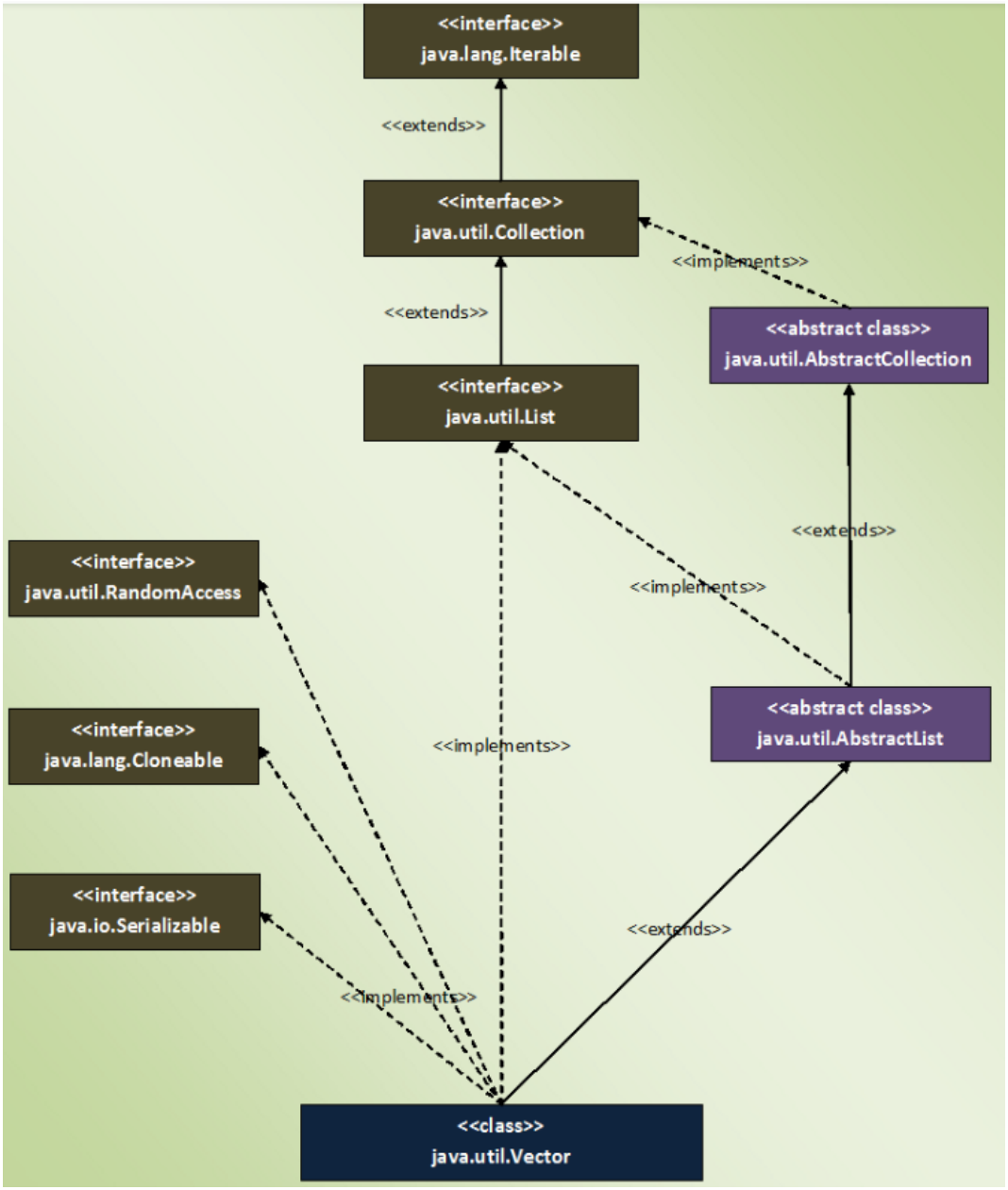


Vector Class

The Vector Class is also a dynamically grow-able and shrink-able collection of objects like an ArrayList class. But, the main difference between ArrayList and Vector is that the Vector class is synchronized. That means, only one thread can enter into a vector object at any moment of time.

Vector class is preferred over ArrayList class when you are developing a multi threaded application. But, precautions need to be taken because vector may reduce the performance of your application as it is thread safety and only one thread is allowed to have object lock at any moment of time and remaining threads have to wait until a thread releases the object lock which is held by it. So, it is always recommended that if you don't need a thread safety environment, it is better to use ArrayList class than the Vector class.

Vector class has same features as ArrayList. Vector class also extends AbstractList class and implements List interface. It also implements 3 marker interfaces – RandomAccess, Cloneable and Serializable



LinkedList Class

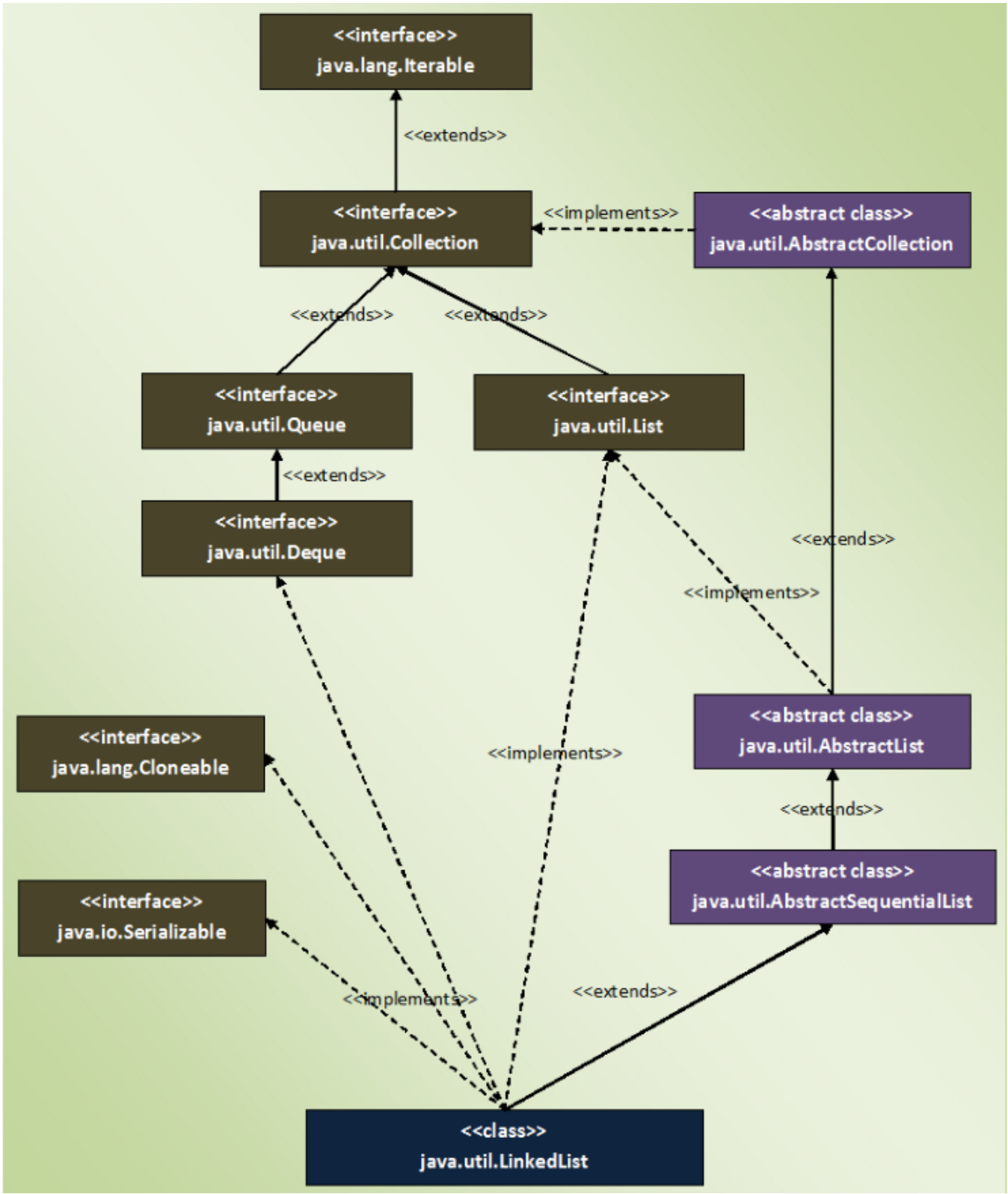
The LinkedList class in Java is an implementation of doubly linked list which can be used both as a List as well as Queue. The LinkedList in java can have any type of elements including null and duplicates. Elements can be inserted and can be removed from both the ends and can be retrieved from any arbitrary position.
The LinkedList class extends AbstractSequentialList and implements List and Deque interfaces. It also implements 2 marker interfaces – Cloneable and Serializable.
Elements in the LinkedList are called Nodes. Where each node consists of three parts – Reference To Previous Element, Value Of The Element and Reference To Next Element.

Insertion and removal operations in LinkedList are faster than ArrayList. Because in LinkedList, there is no need to shift the elements after each insertion and removal. only references of next and previous elements need to be changed.
Retrieval of the elements is very slow in LinkedList as compared to ArrayList. Because in LinkedList, you have to traverse from beginning or end (whichever is closer to the element) to reach the element.
The LinkedList can be used as a stack. It has the methods pop() and push() which make it to function as Stack.The LinkedList can also be used as ArrayList, Queue, SIngle linked list and doubly linked list.LinkedList can have multiple null and duplicate element
LinkedList class in Java is not of type Random Access.

Insertion At Head	Insertion In The Middle	Insertion At Tail
addFirst(E e)	add(int index, E e)	add(E e)
offerFirst(E e)	addAll(int index, Collection c)	addAll(Collection c)
		offer(E e)
		offerLast(E e)

Removing from head	Removing from the middle	Removing from the tail
poll()	Remove(int index)	pollLast()
pollFirst()		removeLast()
remove()		
removeFirst()		

Retrieving from the head	Retrieving from the middle	Retrieving from the tail
element()	get(int index)	getLast()
getFirst()		peekLast()
peek()		
peekFirst()		

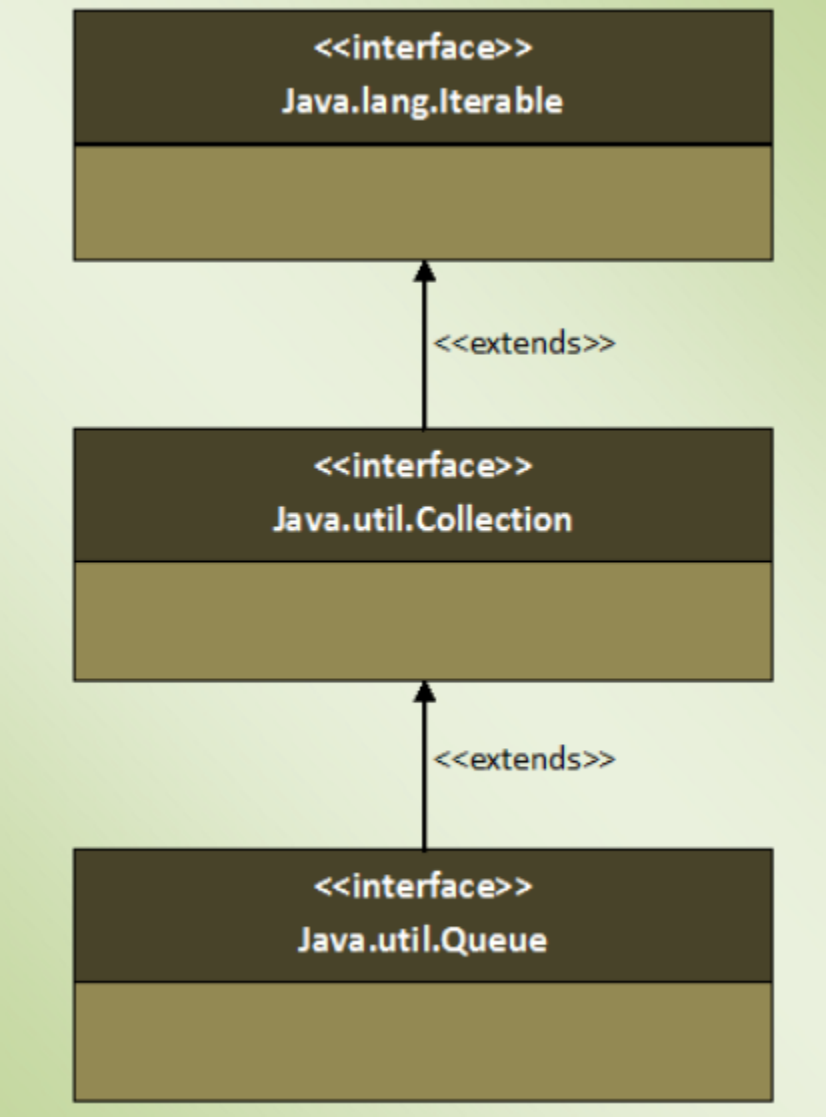

<<interface>>
java.lang.Iterable
<<extends>>
<<interface>>
java.util.Collection
<<implements>>
<<abstract class>>
java.util.AbstractCollection
<<extends>>
<<extends>>
<<interface>>
java.util.Queue
<<interface>>
java.util.List
<<extends>>
<<interface>>
java.util.Deque
<<extends>>
<<implements>>
<<abstract class>>
java.util.AbstractList
<<interface>>
java.lang.Cloneable
<<implements>>
<<extends>>
<<abstract class>>
java.util.AbstractSequentialList
<<interface>>
java.io.Serializable
<<implements>>
<<extends>>
<<class>>
java.util.LinkedList

Queue Interface

The Queue Interface extends Collection interface.Queue is a data structure where elements are added from one end called tail of the queue and elements are removed from another end called head of the queue. Queue is also a first-in-first-out type of data structure (except priority queue). That means an element which is inserted first will be the first element to be removed from the queue. You can't add or get or set elements at an arbitrary position in the queues.

Null elements are not allowed in the queue. Queue can have duplicate elements.Unlike a normal list, queue is not random access. i.e you can't set or insert or get elements at arbitrary positions.

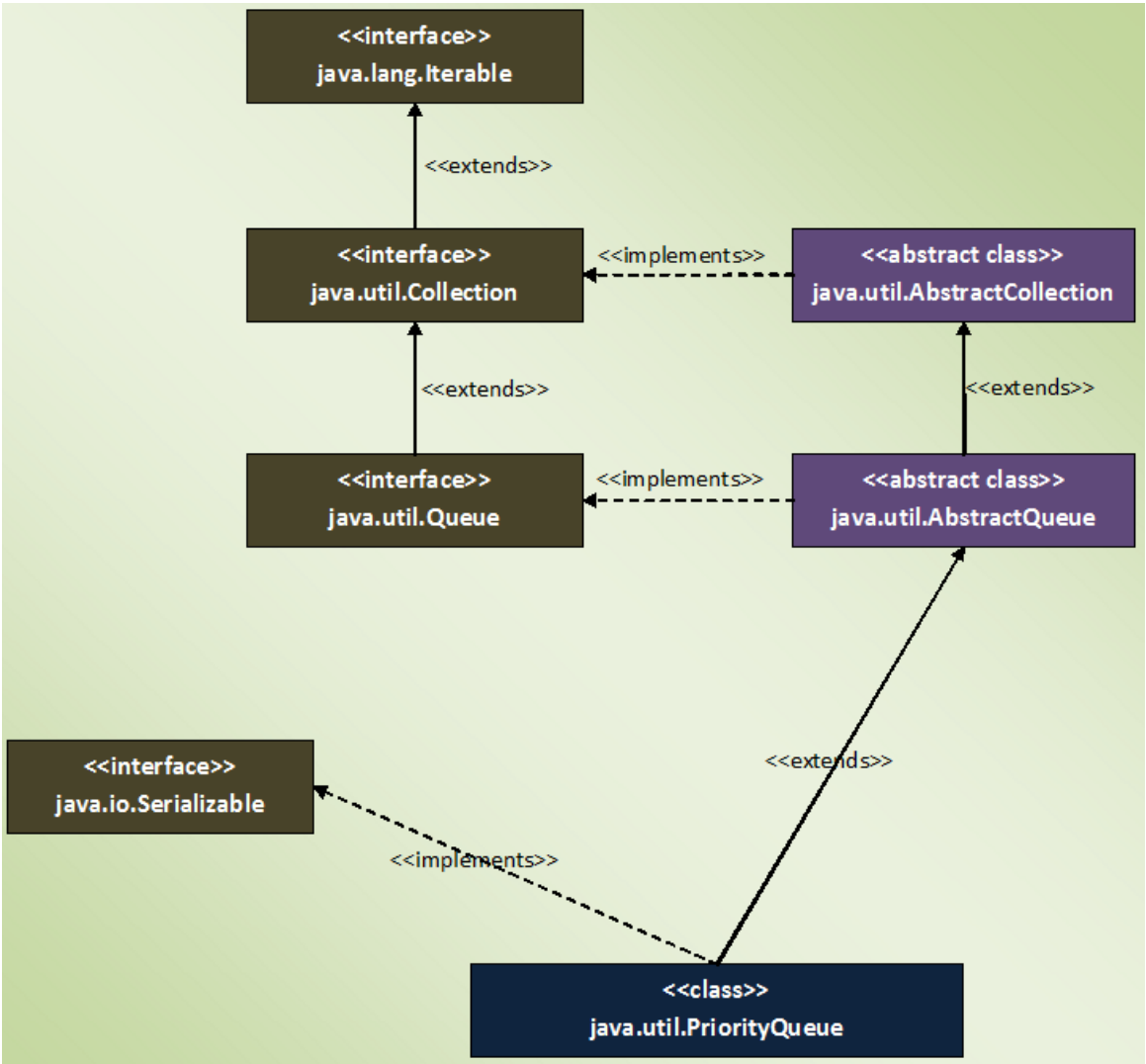
Operation	Throws An Exception If operation is not possible	Returns null or false if operation is not possible
Add an element to the queue.	add()	offer()
Retrieve an element from the head of the queue.	element()	peek()
Retrieve And Remove an element from the head of the queue.	remove()	poll()



PriorityQueue Class

The PriorityQueue is a queue in which elements are ordered according to specified Comparator. You have to specify this Comparator while creating a PriorityQueue itsel. If no Comparator is specified, elements will be placed in their natural order. The PriorityQueue is a special type of queue because it is not a First-In-First-Out (FIFO) as in the normal queues. But, elements are placed according to supplied Comparator.PriorityQueue class extends **AbstractQueue** class which in turn implements **Queue** interface. PriorityQueue also implements one marker interface – **java.io.Serializable** interface. Below is the hierarchy diagram of the PriorityQueue class.

Elements in the PriorityQueue are ordered according to supplied Comparator. If Comparator is not supplied, elements will be placed in their natural order.The PriorityQueue is unbounded. That means the capacity of the PriorityQueue increases automatically if the size exceeds capacity. But, how it grows is not specified.The PriorityQueue can have duplicate elements but can not have null elements.All elements of the PriorityQueue must be of Comparable type. Otherwise ClassCastException will be thrown at run time.The head element of the PriorityQueue is always the least element and tail element is always the largest element according to specified Comparator.The default initial capacity of PriorityQueue is 11.You can retrieve the Comparator used to order the elements of the PriorityQueue using comparator() method.PriorityQueue is not a thread safe.



Deque Interface

The Deque is the short name for “Double Ended Queue“. As the name suggest, Deque is a linear collection of objects which supports insertion and removal of elements from both the ends. The Deque interface defines the methods needed to insert, retrieve and remove the elements from both the ends.The Deque interface is introduced in Java SE 6. It extends Queue interface

The main advantage of Deque is that you can use it as both Queue (FIFO) as well as Stack (LIFO). The Deque interface has all those methods required for FIFO and LIFO operations. Some of those methods throw an exception if operation is not possible and some methods return a special value (null or false) if operation fails.

Unlike Queue, Deque can have null elements. But, it is recommended not to insert null elements as many methods return null to indicate Deque is empty.Deque can have duplicate element

Random access is not possible with the Deque.

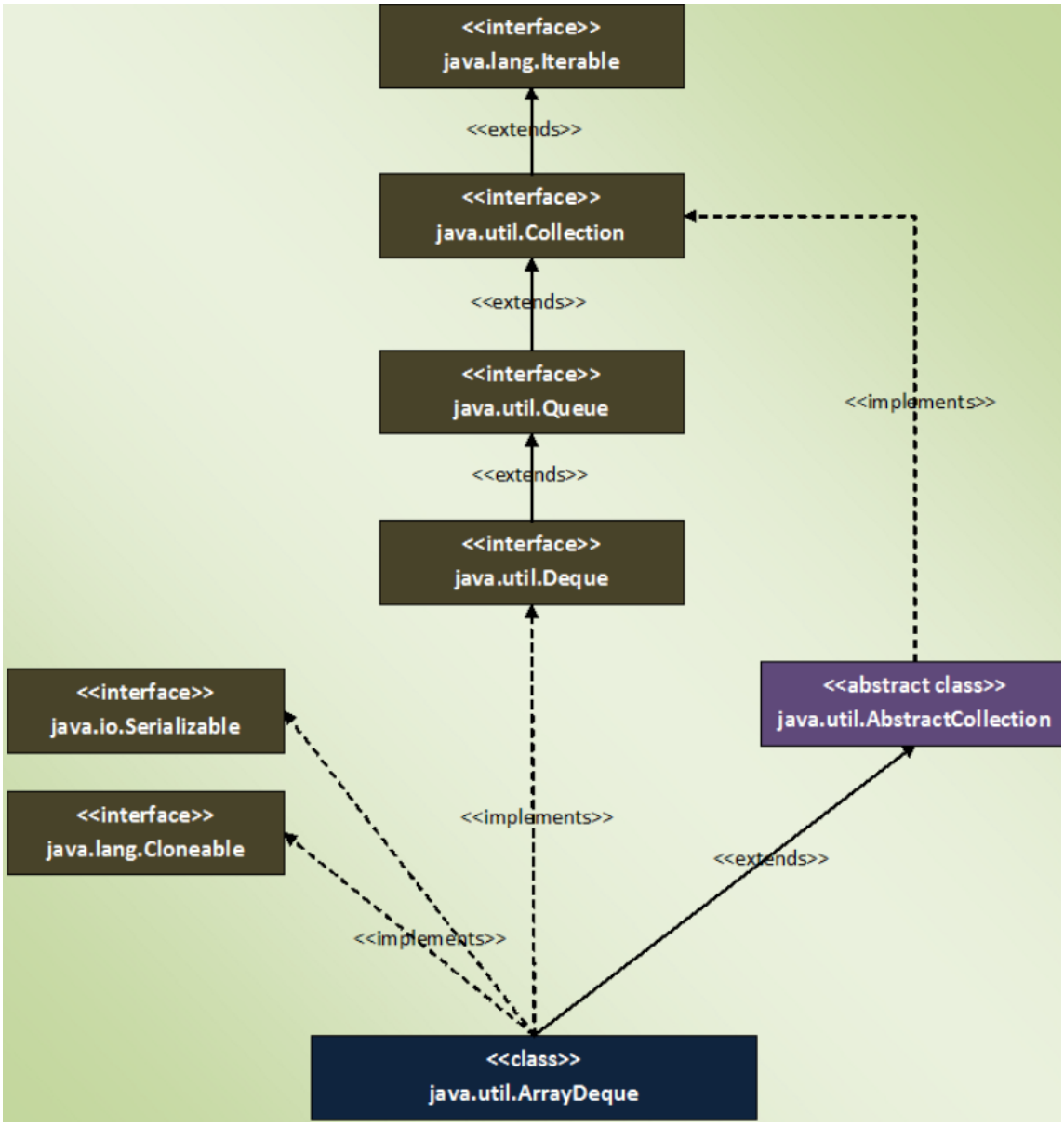
Queue Methods	Equivalent Deque Methods
add()	addLast()
offer()	OfferLast()
element()	getFirst()
peek()	peekFirst()
remove()	removeFirst()
poll()	pollFirst()

Stack Methods	Equivalent Deque Methods
push()	addFirst()
pop()	removeFirst()
peek()	peekFirst()

ArrayDeque Class

The ArrayDeque class in Java is introduced from JDK 1.6. It is an implementation of Deque Interface which allows insertion of elements at both the ends. It does not have any restrictions on capacity. It expands automatically as we add more elements. The ArrayDeque class extends the AbstractCollection class and implements the Deque interface. It also implements Cloneable and Serializable marker interfaces.

ArrayDeque is a resizable-array implementation.It will grow automatically as we add elements.Default initial capacity of ArrayDeque is 16.ArrayDeque can be used as a stack (LIFO) as well as a queue (FIFO). Performance of ArrayDeque is sometimes considered as the best among the collection framework.You can't perform indexed operations on ArrayDeque. ArrayDeque is not a thread safe



Set Interface

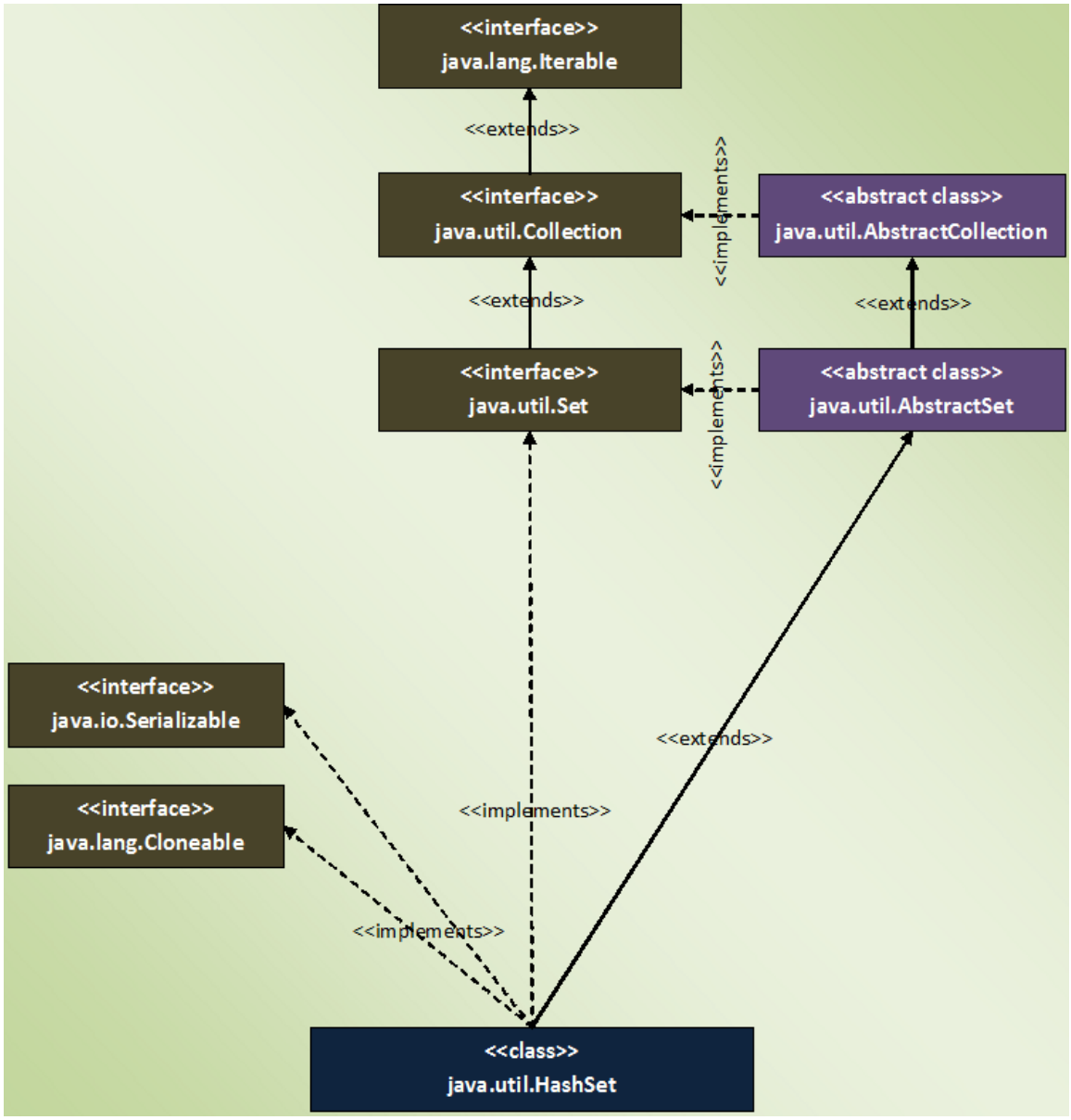
The Set interface defines a set. The set is a linear collection of objects with no duplicates. Duplicate elements are not allowed in a set. The Set interface extends the Collection interface. Set interface does not have its own methods. All its methods are inherited from the Collection interface. The only change that has been made to the Set interface is that add() method will return false if you try to insert an element which is already present in the set.
Set contains only unique elements.Set can contain only one null element.Random access of elements is not possible.Order of elements in a set is implementation dependent. HashSet elements are ordered on the hash code of elements. TreeSet elements are ordered according to supplied Comparator (If no Comparator is supplied, elements will be placed in ascending order) and LinkedHashSet maintains insertion order.

SL No.	Method	Description
1	int size()	Returns the number of elements in the set.
2	boolean isEmpty()	Checks whether the set is empty or not.
3	boolean contains(Object o)	Checks whether this set has specified element.
4	Iterator<E> iterator()	Returns an iterator over the set.
5	Object[] toArray()	It returns an array containing all elements of the set.
6	<T> T[] toArray(T[] a)	It returns an array of specified type containing all elements of this set.
7	boolean add(E e)	This method adds specified element to this set only if that element doesn't present already. It returns true if element is added successfully otherwise returns false.
8	boolean remove(Object o)	Removes the specified element from this set.
9	boolean containsAll(Collection<?> c)	It checks whether this set contains all elements of passed collection.
10	boolean addAll(Collection<? extends E> c)	Adds all elements of the passed collection to this set if they are not already present.
11	boolean removeAll(Collection<?> c)	Removes all elements of this set which are also elements of passed collection.
12	boolean retainAll(Collection<?> c)	Retains only those elements in this set which are also elements of passed collection.
13	void clear()	Removes all elements in this set.
14	boolean equals(Object o)	Compares the specified object with this set for equality.
15	int hashCode()	Returns the hash code value of this set.

HashSet Class

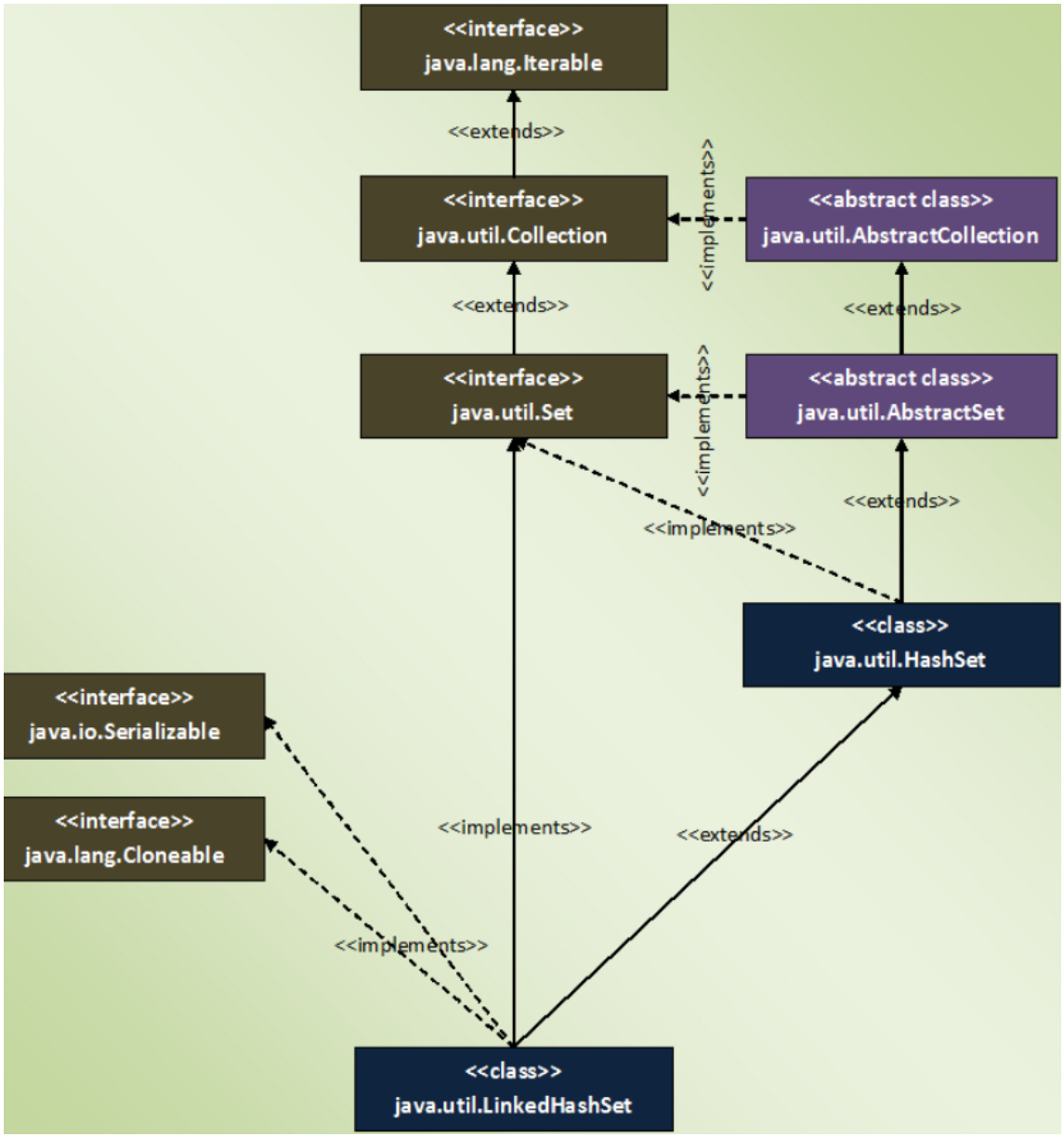
The HashSet class in Java is an implementation of Set interface. HashSet is a collection of objects which contains only unique elements.

HashSet class internally uses HashMap to store the objects. The elements you enter into HashSet will be stored as keys of HashMap and their values will be a constant.HashSet does not allow duplicate elements. If you try to insert a duplicate element, the older element will be overwritten.HashSet can have a maximum one null element.HashSet doesn't maintain any order. HashSet class is not synchronized.HashSet offers constant time performance for insertion, removal and retrieval operations.



LinkedHashSet Class

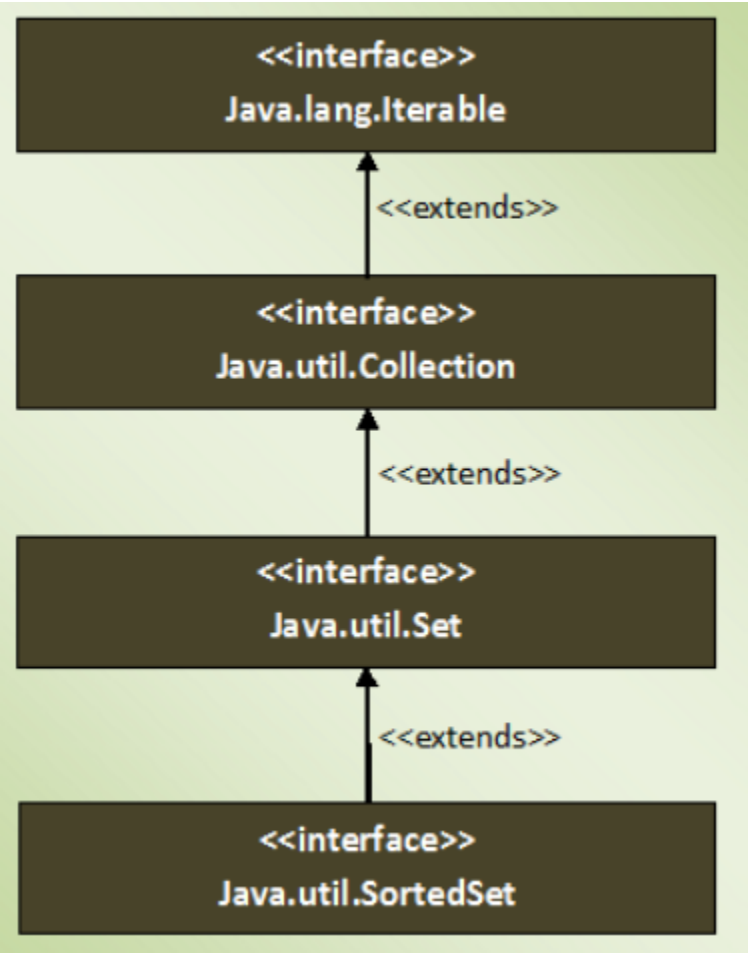
The LinkedHashSet in java is an ordered version of HashSet which internally maintains one doubly linked list running through it's elements. This doubly linked list is responsible for maintaining the insertion order of the elements. Unlike HashSet which maintains no order, LinkedHashSet maintains insertion order of elements. i.e elements are placed in the order they are inserted. LinkedHashSet is recommended over HashSet if you want a unique collection of objects in an insertion order.The LinkedHashSet class extends HashSet class and implements Set interface. It also implements Cloneable and Serializable marker interfaces. Below is the hierarchy diagram of the LinkedHashSet class in java.
LinkedHashSet internally uses LinkedHashMap.LinkedHashSet maintains insertion order.LinkedHashSet is not synchronized



SortedSet Interface

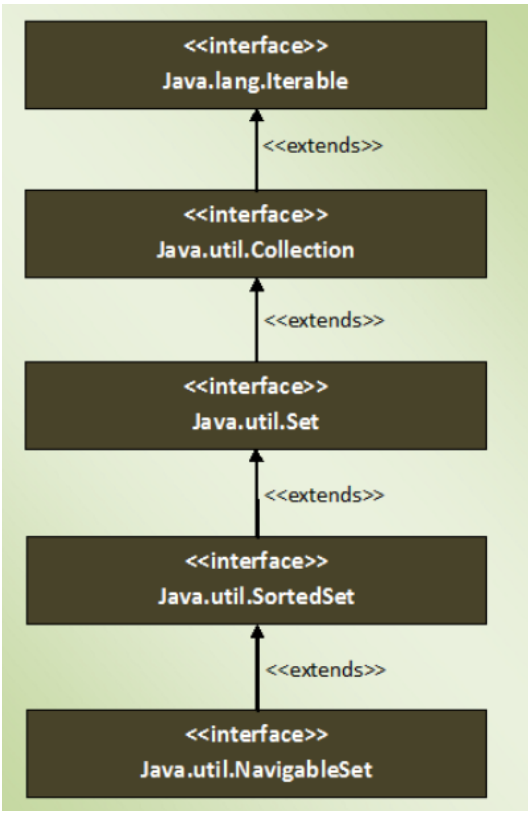
The SortedSet interface extends the Set interface. SortedSet is a set in which elements are placed according to the supplied comparator. This Comparator is supplied while creating a SortedSet. If you don't supply comparator, elements will be placed in ascending order.SortedSet interface defines 6 more methods along with the inherited methods from Set interface SortedSet can not have null , duplicate and Random access elements.SortedSet elements are sorted according to supplied Comparator.Inserted elements must be of Comparable type and they must be mutually Comparable.

	SortedSet Interface Methods	Description
	Comparator<? super E> comparator()	Returns Comparator used to order the elements. If no comparator is supplied, it returns null.
	SortedSet<E> subSet(E fromElement, E toElement)	Returns a portion of this set whose elements range from 'fromElement' (Inclusive) and 'toElement' (Exclusive).
	SortedSet<E> headSet(E toElement)	Returns a SortedSet whose elements are in the range from first element of the set (Inclusive) to 'toElement' (exclusive).
	SortedSet<E> tailSet(E fromElement)	Returns a SortedSet whose elements are in the range from 'fromElement' (Inclusive) to last element of the set (exclusive).
	E first()	Returns first element of the SortedSet.
	E last()	Returns last element of the SortedSet.



NavigableSet Interface

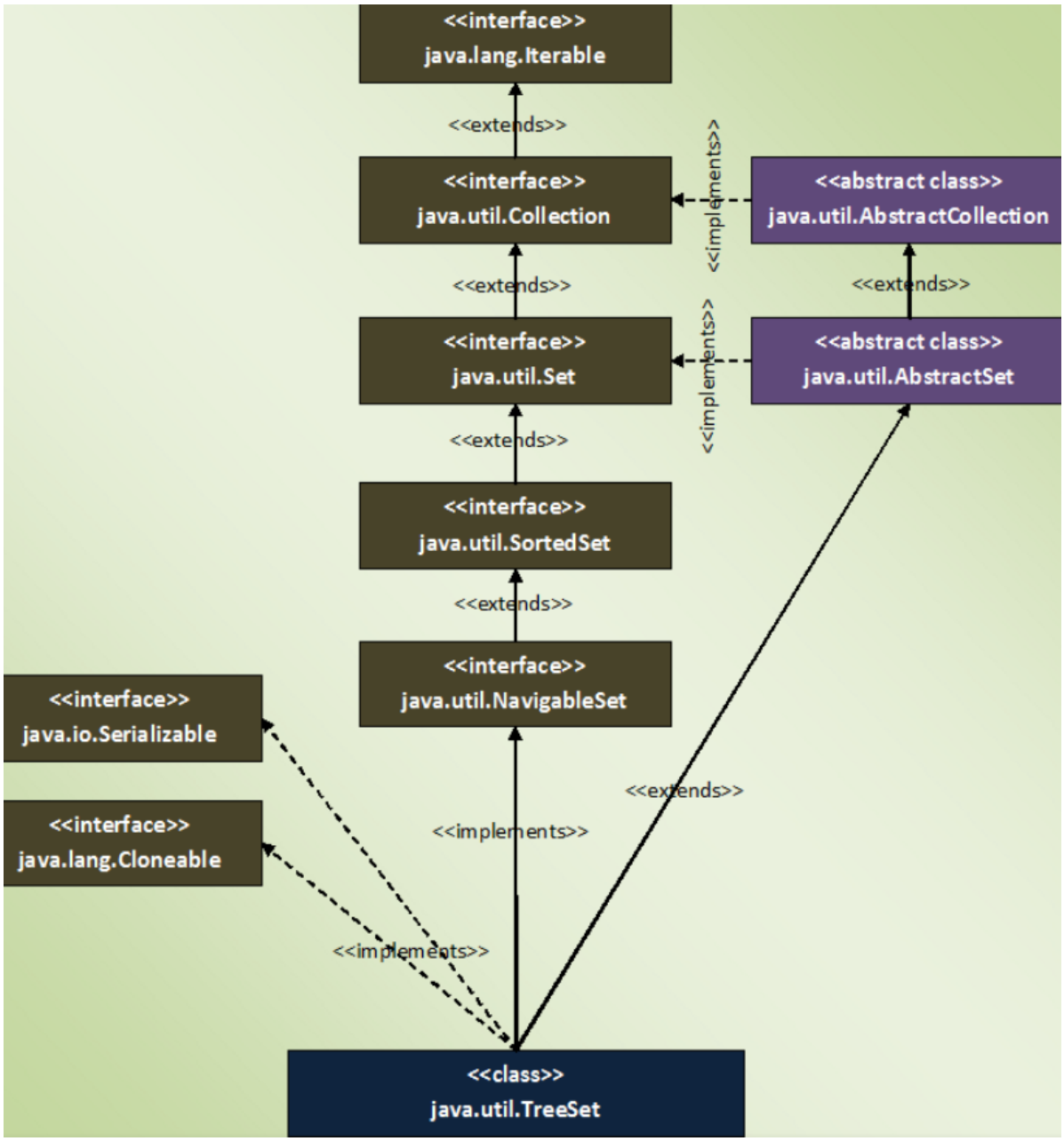
The NavigableSet is a SortedSet with navigation facilities. The NavigableSet interface provides many methods through them you can easily find closest matches of any given element. It has the methods to find out less than, less than or equal to, greater than and greater than or equal of any element in a SortedSet.The NavigableSet interface extends SortedSet interface



NavigableSet Interface Methods	Description
E lower(E e)	Returns greatest element in this set which is strictly less than the given element.
E floor(E e)	Returns greatest element in this set which is less than or equal to the given element.
E ceiling(E e)	Returns the least element in this set which is greater than or equal to the given element.
E higher(E e)	Returns the least element in this set which is strictly greater than the given element.
E pollFirst()	Retrieves and removes the first element in this set.
E pollLast()	Retrieves and removes last element in this set.
NavigableSet<E> descendingSet()	Returns reverse order view of this set.
Iterator<E> descendingIterator()	Returns an iterator over the elements of this set in descending order.
NavigableSet<E> subSet(E fromElement, boolean fromInclusive, E toElement, boolean toInclusive)	Returns a view of this set whose elements are in the range from 'fromElement' to 'toElement'.
NavigableSet<E> headSet(E toElement, boolean inclusive)	Returns a view of this set whose elements are in the range from first element of this set to 'toElement'.
NavigableSet<E> tailSet(E fromElement, boolean inclusive)	Returns a view of this element whose elements are in the range from 'fromElement' to last element of this set.

TreeSet Class

The TreeSet is another popular implementation of Set interface. We have seen other two implementations of Set interface – HashSet and LinkedHashSet. HashSet doesn't maintain any order whereas LinkedHashSet maintains insertion order. The main difference between these two implementations and Treeset is, elements in TreeSet are sorted according to supplied Comparator. You need to supply this Comparator while creating a TreeSet itself. If you don't pass any Comparator while creating a TreeSet, elements will be placed in their natural ascending order.The TreeSet class in java is a direct implementation of NavigableSet interface which in turn extends SortedSet interface (which in turn extends Set interface)

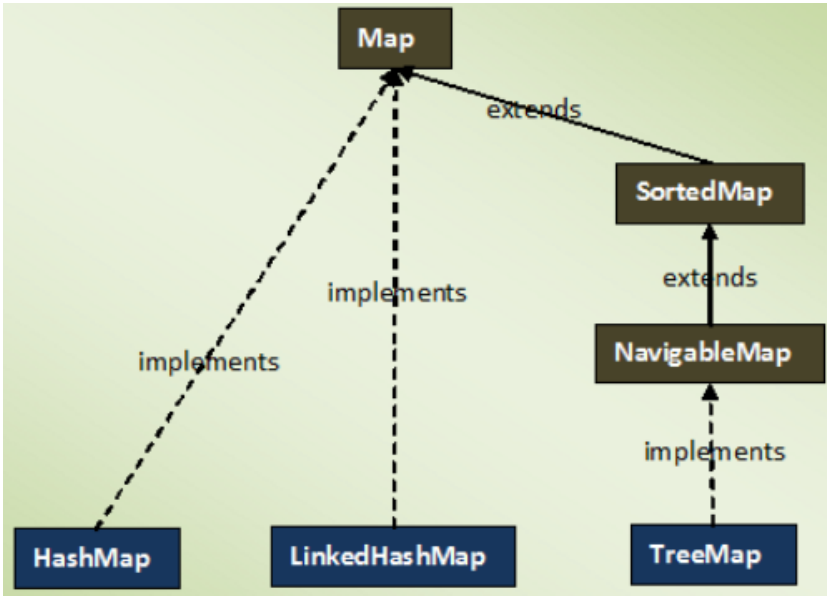


Map Interface

The Map interface in java is one of the four top level interfaces of Java Collection Framework along with List, Set and Queue interfaces. But, unlike others, it doesn't inherit from Collection interface. Instead it starts it's own interface hierarchy for maintaining the key-value associations. Map is an object of key-value pairs where each key is associated with a value. This interface is the replacement for 'Dictionary' class which is an abstract class introduced in JDK 1.0.

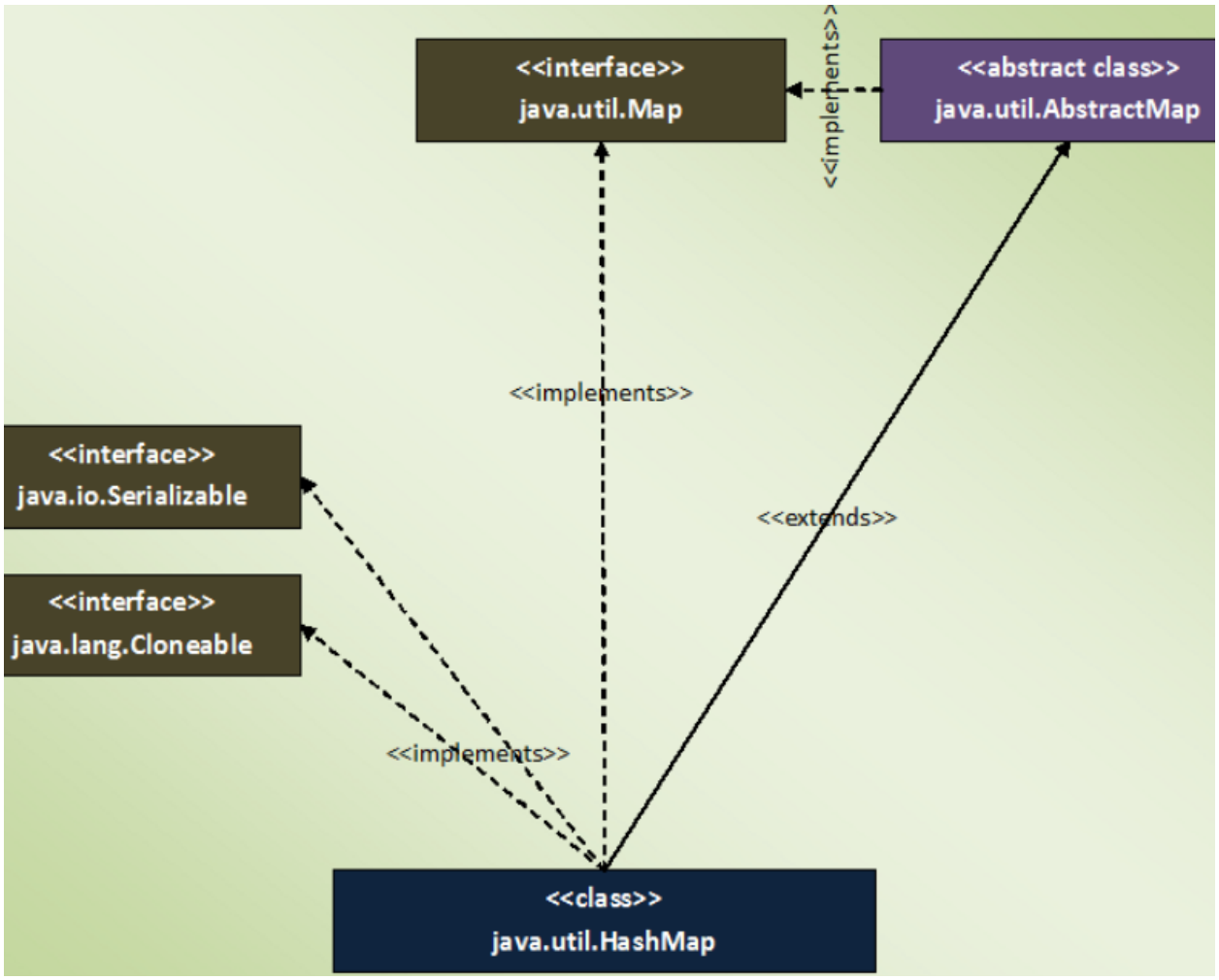
Map interface is a part of Java Collection Framework, but it doesn't inherit Collection Interface. A map can not have duplicate keys but can have duplicate values.Each key at most must be associated with one value.Each key-value pairs of the map are stored as Map.Entry objects. Map.Entry is an inner interface of Map interface. The common implementations of Map interface are HashMap, LinkedHashMap and TreeMap.Order of elements in map is implementation dependent. HashMap doesn't maintain any order of elements. LinkedHashMap maintains insertion order of elements. Where as TreeMap places the elements according to supplied Comparator.

Methods	Descriptions
int size()	Returns number of key-value pairs in this map.
boolean isEmpty()	Checks whether this map is empty or not.
boolean containsKey(Object key)	Returns true if this map contains a mapping for the specified key.
boolean containsValue(Object value)	Returns true if this map contains one or more keys associated with the specified value.
V get(Object key)	Returns value associated with the specified key.
V put(K key, V value)	Adds the specified key-value pair to this map. If the specified key already exist in the map, old value will be replaced by the specified value.
V remove(Object key)	Removes the specified key along with it's value from this map.
void putAll(Map<? extends K, ? extends V> m)	Copies all key-value pairs from the specified map to this map.
void clear()	Removes all mappings from this map.
Set<K> keySet()	Returns a set containing all keys of this map. The returned set is backed by actual map. So, changes made to the map are reflected in the set and vice-versa.
Collection<V> values()	Returns a collection of values of this map. The returned collection is backed by actual map. So, any changes made to the map is reflected in collection and vice-versa.
Set<Map.Entry<K, V>> entrySet()	Returns set view of the mappings contained in this map.
boolean equals(Object o)	Compares the specified object with this map.
int hashCode()	Returns hashcode value of this map.



HashMap Class

The java.util.HashMap is a popular implementation of Map interface which holds the data as key-value pairs. HashMap extends AbstractMap class and implements Cloneable and Serializable interfaces.HashMap is not synchronized. HashMap maintains no order.Default initial capacity of HashMap is 16.



Methods

public V put(K key, V value),public void putAll(Map m),public V get(Object key),public int size(),public boolean isEmpty()

Array Vs ArrayList?
ArrayList Vs Vector?
ArrayList Vs LinkedList?
How HashSet works?
How LinkedHashSet works?
HashSet Vs LinkedHashSet Vs TreeSet?
How HashMap works?
HashMap Vs HashSet?
HashMap Vs HashTable?