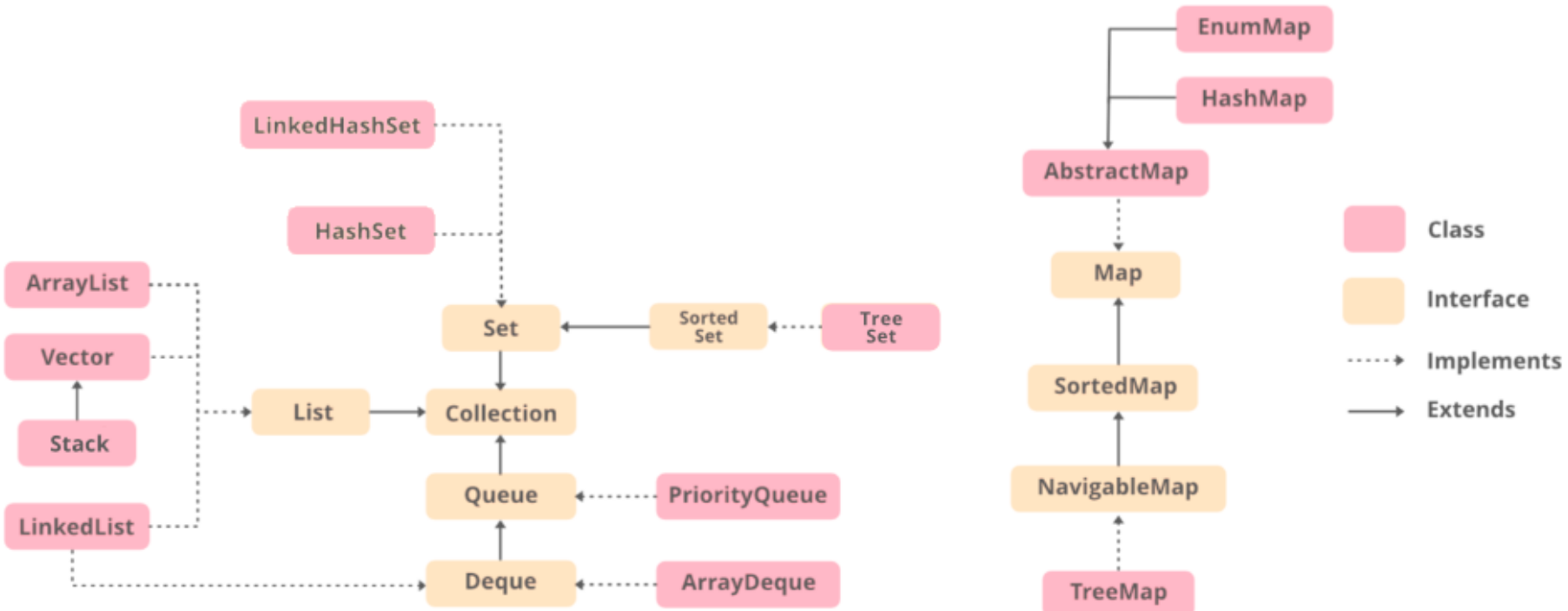
Java Collections



Time complexity

Java Collections APIs	Access	Insert	Delete	Search	Traverse
List					
ArrayList	O(1)	O(1)	O(n)	O(n)	O(n)
LinkedList	O(n)	O(1)	O(n)	O(n)	O(n)
Stack	O(1)	O(1)	O(1)	O(n)	O(n)
Queue					
ArrayDeque	O(1)	O(1)	O(1)	O(n)	O(n)
PriorityQueue	O(logn)	O(logn)	O(logn)	O(logn)	O(n)
Map					
HashMap	O(1)	O(1)	O(1)	O(1)	O(n)
LinkedHashMap	O(1)	O(1)	O(1)	O(1)	O(n)
TreeMap	O(logn)	O(logn)	O(logn)	O(logn)	O(n)
Dictionary					
Hashtable	O(1)	O(1)	O(1)	O(1)	O(n)
Set					
HashSet	O(1)	O(1)	O(1)	O(1)	O(n)
LinkedHashSet	O(1)	O(1)	O(1)	O(1)	O(n)
TreeSet	O(logn)	O(logn)	O(logn)	O(logn)	O(n)

Methods Shortcut with complexity

List	Add	Remove	Get	Contains	Next	Data Structure
ArrayList	O(1)	O(n)	O(1)	O(n)	O(1)	Array
LinkedList	O(1)	O(1)	O(n)	O(n)	O(1)	Linked List
CopyOnWriteArrayList	O(n)	O(n)	O(1)	O(n)	O(1)	Array

Set	Add	Remove	Contains	Next	Size	Data Structure
HashSet	O(1)	O(1)	O(1)	O(h/n)	O(1)	Hash Table
LinkedHashSet	O(1)	O(1)	O(1)	O(1)	O(1)	Hash Table + Linked List
EnumSet	O(1)	O(1)	O(1)	O(1)	O(1)	Bit Vector
TreeSet	O(log n)	O(log n)	O(log n)	O(log n)	O(1)	Redblack tree
CopyOnWriteArraySet	O(n)	O(n)	O(n)	O(1)	O(1)	Array
ConcurrentSkipListSet	O(log n)	O(log n)	O(log n)	O(1)	O(n)	Skip List

Map	Put	Remove	Get	ContainsKey	Next	Data Structure
HashMap	O(1)	O(1)	O(1)	O(1)	O(h / n)	Hash Table
LinkedHashMap	O(1)	O(1)	O(1)	O(1)	O(1)	Hash Table + Linked List
IdentityHashMap	O(1)	O(1)	O(1)	O(1)	O(h / n)	Array
WeakHashMap	O(1)	O(1)	O(1)	O(1)	O(h / n)	Hash Table
EnumMap	O(1)	O(1)	O(1)	O(1)	O(1)	Array
TreeMap	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)	Redblack tree
ConcurrentHashMap	O(1)	O(1)	O(1)	O(1)	O(h / n)	Hash Tables
ConcurrentSkipListMap	O(log n)	O(log n)	O(log n)	O(log n)	O(1)	Skip List

Queue	Offer	Peak	Poll	Remove	Size	Data Structure
PriorityQueue	O(log n)	O(1)	O(log n)	O(n)	O(1)	Priority Heap
LinkedList	O(1)	O(1)	O(1)	O(1)	O(1)	Array
ArrayDequeue	O(1)	O(1)	O(1)	O(n)	O(1)	Linked List
ConcurrentLinkedQueue	O(1)	O(1)	O(1)	O(n)	O(1)	Linked List
ArrayBlockingQueue	O(1)	O(1)	O(1)	O(n)	O(1)	Array
PrioriritryBlockingQueue	O(log n)	O(1)	O(log n)	O(n)	O(1)	Priority Heap
SynchronousQueue	O(1)	O(1)	O(1)	O(n)	O(1)	None
DelayQueue	O(log n)	O(1)	O(log n)	O(n)	O(1)	Priority Heap
LinkedBlockingQueue	O(1)	O(1)	O(1)	O(n)	O(1)	Linked List

List Interface

Array List Class [View Original]

Method	Description
void add(int index, E element)	It is used to insert the specified element at the specified position in a list.
boolean add(E e)	It is used to append the specified element at the end of a list.
boolean addAll(Collection<? extends E> c)	It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
boolean addAll(int index, Collection<? extends E> c)	It is used to append all the elements in the specified collection, starting at the specified position of the list.
void clear()	It is used to remove all of the elements from this list.

E get(int index)	It is used to fetch the element from the particular position of the list.
boolean isEmpty()	It returns true if the list is empty, otherwise false.
Object[] toArray()	It is used to return an array containing all of the elements in this list in the correct order.
<T> T[] toArray(T[] a)	It is used to return an array containing all of the elements in this list in the correct order.
Object clone()	It is used to return a shallow copy of an ArrayList.
boolean contains(Object o)	It returns true if the list contains the specified element.
int indexOf(Object o)	It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
E remove(int index)	It is used to remove the element present at the specified position in the list.
boolean remove(Object o)	It is used to remove the first occurrence of the specified element.
boolean removeAll(Collection<?> c)	It is used to remove all the elements from the list.
void retainAll(Collection<?> c)	It is used to retain all the elements in the list that are present in the specified collection.
E set(int index, E element)	It is used to replace the specified element in the list, present at the specified position.
void sort(Comparator<? super E> c)	It is used to sort the elements of the list on the basis of the specified comparator.
int size()	It is used to return the number of elements present in the list.

Linked List Class View Original

Method	Description
boolean add(E e)	It is used to append the specified element to the end of a list.
void add(int index, E element)	It is used to insert the specified element at the specified position index in a list.
boolean addAll(Collection<? extends E> c)	It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
boolean addAll(Collection<? extends E> c)	It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
boolean addAll(int index, Collection<? extends E> c)	It is used to append all the elements in the specified collection, starting at the specified position of the list.
void addFirst(E e)	It is used to insert the given element at the beginning of a list.
void addLast(E e)	It is used to append the given element to the end of a list.
void clear()	It is used to remove all the elements from a list.
Object clone()	It is used to return a shallow copy of an ArrayList.
boolean contains(Object o)	It is used to return true if a list contains a specified element.
E element()	It is used to retrieve the first element of a list.
E get(int index)	It is used to return the element at the specified position in a list.
E getFirst()	It is used to return the first element in a list.
E getLast()	It is used to return the last element in a list.

int indexOf(Object o)	It is used to return the index in a list of the first occurrence of the specified element, or -1 if the list does not contain any element.
boolean offer(E e)	It adds the specified element as the last element of a list.
boolean offerFirst(E e)	It inserts the specified element at the front of a list.
boolean offerLast(E e)	It inserts the specified element at the end of a list.
E peek()	It retrieves the first element of a list
E peekFirst()	It retrieves the first element of a list or returns null if a list is empty.
E peekLast()	It retrieves the last element of a list or returns null if a list is empty.
E poll()	It retrieves and removes the first element of a list.
E pollFirst()	It retrieves and removes the first element of a list, or returns null if a list is empty.
E pollLast()	It retrieves and removes the last element of a list, or returns null if a list is empty.
E pop()	It pops an element from the stack represented by a list.
void push(E e)	It pushes an element onto the stack represented by a list.
E remove()	It is used to retrieve and removes the first element of a list.
E remove(int index)	It is used to remove the element at the specified position in a list.
boolean remove(Object o)	It is used to remove the first occurrence of the specified element in a list.
E removeFirst()	It removes and returns the first element from a list.
E removeLast()	It removes and returns the last element from a list.
E set(int index, E element)	It replaces the element at the specified position in a list with the specified element.
Object[] toArray()	It is used to return an array containing all the elements in a list in proper sequence (from first to the last element).
<T> T[] toArray(T[] a)	It returns an array containing all the elements in the proper sequence (from first to the last element); the runtime type of the returned array is that of the specified array.
int size()	It is used to return the number of elements in a list.

Queue Interface

Priority Queue Class [View Original]

Method	Description
boolean add(object)	It is used to insert the specified element into this queue and return true upon success.
boolean offer(object)	It is used to insert the specified element into this queue.
Object remove()	It is used to retrieves and removes the head of this queue.
Object poll()	It is used to retrieves and removes the head of this queue, or returns null if this queue is empty.
Object element()	It is used to retrieves, but does not remove, the head of this queue.
Object peek()	It is used to retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.

Set Interface

Hash Set Class View Original

SN	Modifier & Type	Method	Description
1)	boolean	add(E e)	It is used to add the specified element to this set if it is not already present.
2)	void	clear()	It is used to remove all of the elements from the set.
3)	object	clone()	It is used to return a shallow copy of this HashSet instance: the elements themselves are not cloned.
4)	boolean	contains(Object o)	It is used to return true if this set contains the specified element.
5)	boolean	isEmpty()	It is used to return true if this set contains no elements.
6)	Iterator<E>	iterator()	It is used to return an iterator over the elements in this set.
7)	boolean	remove(Object o)	It is used to remove the specified element from this set if it is present.
8)	int	size()	It is used to return the number of elements in the set.

Linked Hash Set Class View Original

SN	Modifier & Type	Method	Description
1)	boolean	add(E e)	It is used to add the specified element to this set if it is not already present.
2)	void	clear()	It is used to remove all of the elements from the set.
3)	object	clone()	It is used to return a shallow copy of this HashSet instance: the elements themselves are not cloned.
4)	boolean	contains(Object o)	It is used to return true if this set contains the specified element.
5)	boolean	isEmpty()	It is used to return true if this set contains no elements.
6)	Iterator<E>	iterator()	It is used to return an iterator over the elements in this set.
7)	boolean	remove(Object o)	It is used to remove the specified element from this set if it is present.
8)	int	size()	It is used to return the number of elements in the set.

Tree Set Class View Original

Method	Description
boolean add(E e)	It is used to add the specified element to this set if it is not already present.
boolean addAll(Collection<? extends E> c)	It is used to add all of the elements in the specified collection to this set.
E ceiling(E e)	It returns the equal or closest greatest element of the specified element from the set, or null there is no such element.

Comparator<? super E> comparator()	It returns a comparator that arranges elements in order.
Iterator descendingIterator()	It is used to iterate the elements in descending order.
NavigableSet descendingSet()	It returns the elements in reverse order.
E floor(E e)	It returns the equal or closest least element of the specified element from the set, or null there is no such element.
SortedSet headSet(E toElement)	It returns the group of elements that are less than the specified element.
NavigableSet headSet(E toElement, boolean inclusive)	It returns the group of elements that are less than or equal to(if, inclusive is true) the specified element.
E higher(E e)	It returns the closest greatest element of the specified element from the set, or null there is no such element.
Iterator iterator()	It is used to iterate the elements in ascending order.
E lower(E e)	It returns the closest least element of the specified element from the set, or null there is no such element.
E pollFirst()	It is used to retrieve and remove the lowest(first) element.
E pollLast()	It is used to retrieve and remove the highest(last) element.
Spliterator spliterator()	It is used to create a late-binding and fail-fast spliterator over the elements.
NavigableSet subSet(E fromElement, boolean fromInclusive, E toElement, boolean toInclusive)	It returns a set of elements that lie between the given range.
SortedSet subSet(E fromElement, E toElement))	It returns a set of elements that lie between the given range which includes fromElement and excludes toElement.
SortedSet tailSet(E fromElement)	It returns a set of elements that are greater than or equal to the specified element.
NavigableSet tailSet(E fromElement, boolean inclusive)	It returns a set of elements that are greater than or equal to (if, inclusive is true) the specified element.
boolean contains(Object o)	It returns true if this set contains the specified element.
boolean isEmpty()	It returns true if this set contains no elements.
boolean remove(Object o)	It is used to remove the specified element from this set if it is present.
void clear()	It is used to remove all of the elements from this set.
Object clone()	It returns a shallow copy of this TreeSet instance.
E first()	It returns the first (lowest) element currently in this sorted set.
E last()	It returns the last (highest) element currently in this sorted set.
int size()	It returns the number of elements in this set.

Map Interface

Hash Map Class [View Original]

Method	Description

void clear()	It is used to remove all of the mappings from this map.
boolean isEmpty()	It is used to return true if this map contains no key-value mappings.
Object clone()	It is used to return a shallow copy of this HashMap instance: the keys and values themselves are not cloned.
Set entrySet()	It is used to return a collection view of the mappings contained in this map.
Set keySet()	It is used to return a set view of the keys contained in this map.
V put(Object key, Object value)	It is used to insert an entry in the map.
void putAll(Map map)	It is used to insert the specified map in the map.
V putIfAbsent(K key, V value)	It inserts the specified value with the specified key in the map only if it is not already specified.
V remove(Object key)	It is used to delete an entry for the specified key.
boolean remove(Object key, Object value)	It removes the specified values with the associated specified keys from the map.
V compute(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)	It is used to compute a mapping for the specified key and its current mapped value (or null if there is no current mapping).
V computeIfAbsent(K key, Function<? super K,? extends V> mappingFunction)	It is used to compute its value using the given mapping function, if the specified key is not already associated with a value (or is mapped to null), and enters it into this map unless null.
V computeIfPresent(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)	It is used to compute a new mapping given the key and its current mapped value if the value for the specified key is present and non-null.
boolean containsValue(Object value)	This method returns true if some value equal to the value exists within the map, else return false.
boolean containsKey(Object key)	This method returns true if some key equal to the key exists within the map, else return false.
boolean equals(Object o)	It is used to compare the specified Object with the Map.
void forEach(BiConsumer<? super K,? super V> action)	It performs the given action for each entry in the map until all entries have been processed or the action throws an exception.
V get(Object key)	This method returns the object that contains the value associated with the key.
V getOrDefault(Object key, V defaultValue)	It returns the value to which the specified key is mapped, or defaultValue if the map contains no mapping for the key.
boolean isEmpty()	This method returns true if the map is empty; returns false if it contains at least one key.
V merge(K key, V value, BiFunction<? super V,? super V,? extends V> remappingFunction)	If the specified key is not already associated with a value or is associated with null, associates it with the given non-null value.
V replace(K key, V value)	It replaces the specified value for a specified key.
boolean replace(K key, V oldValue, V newValue)	It replaces the old value with the new value for a specified key.
void replaceAll(BiFunction<? super K,? super V,? extends V> function)	It replaces each entry's value with the result of invoking the given function on that entry until all entries have been processed or the function throws an exception.
Collection<V> values()	It returns a collection view of the values contained in the map.

int size()	This method returns the number of entries in the map.

Linked Hash Map Class [View Original]

Method	Description
V get(Object key)	It returns the value to which the specified key is mapped.
void clear()	It removes all the key-value pairs from a map.
boolean containsValue(Object value)	It returns true if the map maps one or more keys to the specified value.
Set<Map.Entry<K,V>> entrySet()	It returns a Set view of the mappings contained in the map.
void forEach(BiConsumer<? super K,? super V> action)	It performs the given action for each entry in the map until all entries have been processed or the action throws an exception.
V getOrDefault(Object key, V defaultValue)	It returns the value to which the specified key is mapped or defaultValue if this map contains no mapping for the key.
Set<K> keySet()	It returns a Set view of the keys contained in the map
protected boolean removeEldestEntry(Map.Entry<K,V> eldest)	It returns true on removing its eldest entry.
void replaceAll(BiFunction<? super K,? super V,? extends V> function)	It replaces each entry's value with the result of invoking the given function on that entry until all entries have been processed or the function throws an exception.
Collection<V> values()	It returns a Collection view of the values contained in this map.

Tree Map Class [View Original]

Method	Description
Map.Entry<K,V> ceilingEntry(K key)	It returns the key-value pair having the least key, greater than or equal to the specified key, or null if there is no such key.
K ceilingKey(K key)	It returns the least key, greater than the specified key or null if there is no such key.
void clear()	It removes all the key-value pairs from a map.
Object clone()	It returns a shallow copy of TreeMap instance.
Comparator<? super K> comparator()	It returns the comparator that arranges the key in order, or null if the map uses the natural ordering.
NavigableSet<K> descendingKeySet()	It returns a reverse order NavigableSet view of the keys contained in the map.
NavigableMap<K,V> descendingMap()	It returns the specified key-value pairs in descending order.
Map.Entry firstEntry()	It returns the key-value pair having the least key.
Map.Entry<K,V> floorEntry(K key)	It returns the greatest key, less than or equal to the specified key, or null if there is no such key.
void forEach(BiConsumer<? super K,? super V> action)	It performs the given action for each entry in the map until all entries have been processed or the action throws an exception.
SortedMap<K,V> headMap(K toKey)	It returns the key-value pairs whose keys are strictly less than toKey.

NavigableMap<K,V> headMap(K toKey, boolean inclusive)	It returns the key-value pairs whose keys are less than (or equal to if inclusive is true) toKey.
Map.Entry<K,V> higherEntry(K key)	It returns the least key strictly greater than the given key, or null if there is no such key.
K higherKey(K key)	It is used to return true if this map contains a mapping for the specified key.
Set keySet()	It returns the collection of keys exist in the map.
Map.Entry<K,V> lastEntry()	It returns the key-value pair having the greatest key, or null if there is no such key.
Map.Entry<K,V> lowerEntry(K key)	It returns a key-value mapping associated with the greatest key strictly less than the given key, or null if there is no such key.
K lowerKey(K key)	It returns the greatest key strictly less than the given key, or null if there is no such key.
NavigableSet<K> navigableKeySet()	It returns a NavigableSet view of the keys contained in this map.
Map.Entry<K,V> pollFirstEntry()	It removes and returns a key-value mapping associated with the least key in this map, or null if the map is empty.
Map.Entry<K,V> pollLastEntry()	It removes and returns a key-value mapping associated with the greatest key in this map, or null if the map is empty.
V put(K key, V value)	It inserts the specified value with the specified key in the map.
void putAll(Map<? extends K,? extends V> map)	It is used to copy all the key-value pair from one map to another map.
V replace(K key, V value)	It replaces the specified value for a specified key.
boolean replace(K key, V oldValue, V newValue)	It replaces the old value with the new value for a specified key.
void replaceAll(BiFunction<? super K,? super V,? extends V> function)	It replaces each entry's value with the result of invoking the given function on that entry until all entries have been processed or the function throws an exception.
NavigableMap<K,V> subMap(K fromKey, boolean fromInclusive, K toKey, boolean toInclusive)	It returns key-value pairs whose keys range from fromKey to toKey.
SortedMap<K,V> subMap(K fromKey, K toKey)	It returns key-value pairs whose keys range from fromKey, inclusive, to toKey, exclusive.
SortedMap<K,V> tailMap(K fromKey)	It returns key-value pairs whose keys are greater than or equal to fromKey.
NavigableMap<K,V> tailMap(K fromKey, boolean inclusive)	It returns key-value pairs whose keys are greater than (or equal to, if inclusive is true) fromKey.
boolean containsKey(Object key)	It returns true if the map contains a mapping for the specified key.
boolean containsValue(Object value)	It returns true if the map maps one or more keys to the specified value.
K firstKey()	It is used to return the first (lowest) key currently in this sorted map.
V get(Object key)	It is used to return the value to which the map maps the specified key.
K lastKey()	It is used to return the last (highest) key currently in the sorted map.
V remove(Object key)	It removes the key-value pair of the specified key from the map.
Set<Map.Entry<K,V>> entrySet()	It returns a set view of the mappings contained in the map.
int size()	It returns the number of key-value pairs exists in the hashtable.

Collection values()	It returns a collection view of the values contained in the map.

All Important Links

1. ArrayList class
2. LinkedList class
3. List interface
4. HashSet class
5. LinkedHashSet class
6. TreeSet class
7. PriorityQueue class
8. Map interface
9. HashMap class
10. LinkedHashMap class
11. TreeMap class
12. Hashtable class
13. Sorting
14. Comparable interface
15. Comparator interface
16. Properties class in Java