

# JAVA COLLECTION INTERFACES HIERARCHY WITH METHODS

**Map<K, V>**  
V get(Object key)  
V put(K key, V value)  
void putAll(Map<? extends K, ? extends V> m)  
void clear()  
V remove(Object key)  
Set<K> keySet()  
Collection<V> values()  
Set<Map.Entry<K, V>> entrySet()  
int size()  
int hashCode()  
boolean isEmpty()  
boolean containsKey(Object key)  
boolean containsValue(Object value)  
boolean equals(Object o)

**SortedMap<K, V>**  
K firstKey()  
K lastKey()  
Set<K> keySet()  
Collection<V> values()  
Set<Map.Entry<K, V>> entrySet()  
SortedMap<E> headMap(K toKey)  
SortedMap<E> tailMap(K fromKey)  
SortedMap<E> subMap(K fromKey, K toKey)  
Comparator<? super K> comparator()

**NavigableMap<K, V>**  
Map.Entry<K, V> ceilingEntry(K k)  
Map.Entry<K, V> floorEntry(K k)  
Map.Entry<K, V> higherEntry(K k)  
Map.Entry<K, V> lowerEntry(K k)  
Map.Entry<K, V> firstEntry(K k)  
Map.Entry<K, V> lastEntry(K k)  
K ceilingKey(K key)  
K floorKey(K k)  
K lowerKey(K k)  
K higherKey(K k)  
Map.Entry<K, V> pollFirstEntry(K k)  
Map.Entry<K, V> pollLastEntry(K k)  
NavigableSet<K> descendingKeySet()  
NavigableSet<K> navigableKeySet()  
NavigableMap<K, V> descendingMap()  
SortedMap<E> headMap(K toKey)  
SortedMap<E> tailMap(K fromKey)  
SortedMap<E> subMap(K fromKey, K toKey)  
NavigableMap<K, V> headMap(K toKey, boolean inclusive)  
NavigableMap<K, V> tailMap(K fromKey, boolean inclusive)  
NavigableMap<K, V> subMap(K fromKey, boolean inclusive, K toKey, boolean inclusive)

**ConcurrentNavigableMap<K, V>**  
NavigableSet<K> keySet()  
NavigableSet<K> descendingKeySet()  
NavigableSet<K> navigableKeySet()  
ConcurrentNavigableMap<K, V> descendingMap()  
ConcurrentNavigableMap<K, V> headMap(K toKey)  
ConcurrentNavigableMap<K, V> headMap(K toKey, boolean inclusive)  
ConcurrentNavigableMap<K, V> tailMap(K fromKey)  
ConcurrentNavigableMap<K, V> tailMap(K fromKey, boolean inclusive)  
ConcurrentNavigableMap<K, V> subMap(K fromKey, K toKey)  
ConcurrentNavigableMap<K, V> subMap(K fromKey, boolean inclusive, K toKey, boolean inclusive)

**List<E>**  
boolean add(E e)  
void add(int index, E element)  
boolean addAll(Collection<? extends E> c)  
boolean addAll(int index, Collection<? extends E> c)  
E get(int index)  
E set(int index, E element)  
E remove(int index)  
boolean remove(Object o)  
boolean removeAll(Collection<?> c)  
boolean retainAll(Collection<?> c)  
void clear()  
List<E> sublist(int fromIndex, int toIndex)  
int size()  
int hashCode()  
int indexOf(Object o)  
int lastIndexOf(Object o)  
Iterator<E> iterator()  
ListIterator<E> listIterator()  
ListIterator<E> listIterator(int index)  
boolean contains(Object o)  
boolean containsAll(Collection<?> c)  
boolean equals(Object o)  
boolean isEmpty()

**ConcurrentMap<K, V>**  
V putIfAbsent(K key, V value)  
V replace(K key, V value)  
boolean replace(K key, V oldValue, V newValue)  
boolean remove(Object key, Object value)

**Collection<E>**  
boolean add(E e)  
boolean addAll(Collection<?> c)  
void clear()  
boolean remove(Object o)  
boolean removeAll(Collection<?> c)  
Object[] toArray()  
<T> T[] toArray(T[] a)  
int size()  
int hashCode()  
boolean equals()  
boolean isEmpty()  
boolean contains(Object o)  
boolean containsAll(Collection<?> c)  
boolean retainAll(Collection<?> c)

**Set<E>**  
same methods as Collection<E>

**SortedSet<E>**  
E first()  
E last()  
SortedSet<E> headSet(E toElement)  
SortedSet<E> tailSet(E fromElement)  
SortedSet<E> subSet(E fromElement, E toElement)  
Comparator<? super E> comparator()

**NavigableSet<E>**  
E ceiling(E e)  
E floor(E e)  
E lower(E e)  
E higher(E e)  
E pollFirst(E e)  
E pollLast(E e)  
NavigableSet<E> descendingSet()  
SortedSet<E> headSet(E toElement)  
NavigableSet<E> tailSet(E fromElement)  
NavigableSet<E> subSet(E fromElement, E toElement)  
NavigableSet<E> headSet(E toElement, boolean inclusive)  
NavigableSet<E> tailSet(E fromElement, boolean inclusive)  
NavigableSet<E> subSet(E fromElement, boolean inclusive, E toElement, boolean inclusive)  
Iterator<E> descendingIterator()

**Iterable<T>**  
Iterator<T> iterator()

**Iterator<E>**  
boolean hasNext()  
E next()  
void remove()

**ListIterator<E>**  
void add(E e)  
void set(E e)  
boolean hasPrevious()  
E previous()  
int nextIndex()  
int previousIndex()

**Queue<E>**  
boolean add(E e)  
boolean offer(E e)  
E element()  
E peek()  
E poll()  
E remove()

**BlockingQueue<E>**  
boolean add(E e)  
boolean offer(E e)  
boolean offer(E e, long timeout, TimeUnit unit)  
void put(E e)  
E poll(long timeout, TimeUnit unit)  
E take()  
boolean remove(Object o)  
int drainTo(Collection<? super E> c)  
int drainTo(Collection<? super E> c, int maxElements)  
int remainingCapacity()  
boolean contains(Object o)

**TransferQueue<E>**  
void transfer(E e)  
boolean tryTransfer(E e)  
boolean tryTransfer(E e, long timeout, TimeUnit unit)  
int getWaitingConsumerCount()  
boolean hasWaitingConsumer()

**Deque<E>**  
boolean add(E e)  
void addFirst(E e)  
void addLast(E e)  
boolean offer(E e)  
boolean offerFirst(E e)  
boolean offerLast(E e)  
void push(E e)  
E element()  
E getFirst()  
E getLast()  
E peek()  
E peekFirst()  
E peekLast()  
E pop()  
E remove()  
E remove(Object o)  
E removeFirst()  
E removeLast()  
E poll()  
E pollFirst()  
E pollLast()  
boolean removeFirstOccurrence(Object o)  
boolean removeLastOccurrence(Object o)  
boolean contains(Object o)  
Iterator<E> iterator()  
Iterator<E> descendingIterator()

**BlockingDeque<E>**  
boolean add(E e)  
void addFirst(E e)  
void addLast(E e)  
boolean offer(E e)  
boolean offerFirst(E e)  
boolean offerLast(E e)  
boolean offer(E e, long timeout, TimeUnit unit)  
boolean offerFirst(E e, long timeout, TimeUnit unit)  
boolean offerLast(E e, long timeout, TimeUnit unit)  
void push(E e)  
void put(E e)  
void putFirst(E e)  
void putLast(E e)  
E element()  
E getFirst()  
E getLast()  
E peek()  
E peekFirst()  
E peekLast()  
E pop()  
E remove()  
E remove(Object o)  
E removeFirst()  
E removeLast()  
E poll()  
E pollFirst()  
E pollLast()  
E poll(long timeout, TimeUnit unit)  
E pollFirst(long timeout, TimeUnit unit)  
E pollLast(long timeout, TimeUnit unit)  
boolean removeFirstOccurrence(Object o)  
boolean removeLastOccurrence(Object o)  
E take()  
E takeFirst()  
E takeLast()  
boolean contains(Object o)  
Iterator<E> iterator()  
Iterator<E> descendingIterator()