**point-wise list of common methods for various Java collection classes, including Stack, ArrayList, LinkedList, and Map interfaces like HashMap and TreeMap**

**1. Stack methods (java.util.Stack)**

* - void push(E item) // Pushes an item onto the top of the stack
* - E pop() // Removes and returns the item at the top of the stack
* - E peek() // Returns the item at the top of the stack without removing it
* - boolean empty() // Checks if the stack is empty
* - int search(Object o) // Returns the 1-based position of an object in the stack

**2. ArrayList methods (java.util.ArrayList)**

* - void add(E e) // Appends the specified element to the end of the list
  + void add(int index, E element) // Inserts the specified element at the specified position
* - E get(int index) // Returns the element at the specified position
* - E set(int index, E element) // Replaces the element at the specified position
* - E remove(int index) // Removes the element at the specified position
* - boolean remove(Object o) // Removes the first occurrence of the specified element
* - boolean contains(Object o) // Checks if the list contains the specified element
* - int size() // Returns the number of elements in the list
* - boolean isEmpty() // Checks if the list is empty
* - void clear() // Removes all elements from the list
* - Object[] toArray() // Returns an array containing all elements
* - <T> T[] toArray(T[] a) // Returns an array containing all elements of the list
* - int indexOf(Object o) // Returns the index of the first occurrence of the specified element
* - int lastIndexOf(Object o) // Returns the index of the last occurrence of the specified element
  + List<E> subList(int fromIndex, int toIndex) // Returns a view of the portion of the list

**2.1 Additional**

* + void ensureCapacity(int minCapacity) // Increases the capacity of the ArrayList to ensure it can hold at least the number of elements
* - void trimToSize() // Trims the capacity of the ArrayList to the current size
  + boolean removeAll(Collection<?> c) // Removes all elements in this list that are contained in the specified collection
  + boolean retainAll(Collection<?> c) // Retains only the elements in this list that are contained in the specified collection
  + ListIterator<E> listIterator(int index) // Returns a list iterator starting at the specified position

**3 LinkedList methods (java.util.LinkedList)**

* - void add(E e) // Appends the specified element to the end
  + void add(int index, E element) // Inserts the specified element at the specified position
* - E get(int index) // Returns the element at the specified position
* - E set(int index, E element) // Replaces the element at the specified position
* - E remove(int index) // Removes the element at the specified position
* - boolean remove(Object o) // Removes the first occurrence of the specified element
* - boolean contains(Object o) // Checks if the list contains the specified element
* - int size() // Returns the number of elements in the list
* - boolean isEmpty() // Checks if the list is empty
* - void clear() // Removes all elements from the list
* - E getFirst() // Returns the first element
* - E getLast() // Returns the last element
* - E removeFirst() // Removes and returns the first element
* - E removeLast() // Removes and returns the last element
* - void addFirst(E e) // Inserts the specified element at the beginning
* - void addLast(E e) // Appends the specified element to the end
* - Iterator<E> iterator() // Returns an iterator over the elements
* - ListIterator<E> listIterator() // Returns a list iterator over the elements

**3.1 Additional**

* + void addAll(Collection<? extends E> c) // Appends all elements in the specified collection to the end
  + void addAll(int index, Collection<? extends E> c) // Inserts all elements in the specified collection at the specified position
  + boolean removeFirstOccurrence(Object o) // Removes the first occurrence of the specified element

boolean removeLastOccurrence(Object o) // Removes the last occurrence of the specified element

* - boolean offer(E e) // Adds the specified element as the tail (last element)
* - E poll() // Retrieves and removes the head (first element), or returns null if this list is empty
* - E peek() // Retrieves, but does not remove, the head (first element), or returns null if this list is empty

**4 Map methods (java.util.Map)**

* - V put(K key, V value) // Associates the specified value with the specified key
* - V get(Object key) // Returns the value to which the specified key is mapped
* - V remove(Object key) // Removes the mapping for the specified key
  + boolean containsKey(Object key) // Checks if the map contains the specified key
  + boolean containsValue(Object value) // Checks if the map maps one or more keys to the specified value
* - int size() // Returns the number of key-value pairs
* - boolean isEmpty() // Checks if the map is empty
* - void clear() // Removes all key-value pairs
* - Set<K> keySet() // Returns a set view of the keys
* - Collection<V> values() // Returns a collection view of the values
  + Set<Map.Entry<K,V>> entrySet() // Returns a set view of the mappings

**5 HashMap methods (java.util.HashMap)**

* - V put(K key, V value) // Associates the specified value with the specified key
* - V get(Object key) // Returns the value associated with the key
* - V remove(Object key) // Removes the mapping for the specified key
  + boolean containsKey(Object key) // Checks if the map contains the specified key
  + boolean containsValue(Object value) // Checks if the map maps one or more keys to the specified value
* - int size() // Returns the number of key-value pairs
* - boolean isEmpty() // Checks if the map is empty
* - void clear() // Removes all key-value pairs
* - Set<K> keySet() // Returns a set view of the keys
* - Collection<V> values() // Returns a collection view of the values
  + Set<Map.Entry<K,V>> entrySet() // Returns a set view of the mappings
  + V getOrDefault(Object key, V defaultValue) // Returns the value associated with the key, or defaultValue
* - V putIfAbsent(K key, V value) // Adds a key-value pair if the key is not already associated with a value

**5.1 Additional**

* V putAll(Map<? extends K, ? extends V> m) // Copies all of the mappings from the specified map to this map
* - V replace(K key, V value) // Replaces the entry for a key only if currently mapped to some value
* boolean replace(K key, V oldValue, V newValue) // Replaces the entry for a key only if currently mapped to the specified old value
* void forEach(BiConsumer<? super K, ? super V> action) // Performs the given action for each entry in this map until all entries have been processed
* V computeIfAbsent(K key, Function<? super K, ? extends V> mappingFunction) // Computes a value for the given key if it is not already associated with a value
* V computeIfPresent(K key, BiFunction<? super K, ? super V, ? extends V> remappingFunction) // Computes a new mapping for the specified key if it is present

**6 TreeMap methods (java.util.TreeMap)**

* - V put(K key, V value) // Associates the specified value with the specified key
* - V get(Object key) // Returns the value associated with the key
* - V remove(Object key) // Removes the mapping for the specified key
  + boolean containsKey(Object key) // Checks if the map contains the specified key
  + boolean containsValue(Object value) // Checks if the map maps one or more keys to the specified value
* - int size() // Returns the number of key-value pairs
* - boolean isEmpty() // Checks if the map is empty
* - void clear() // Removes all key-value pairs
* - Set<K> keySet() // Returns a set view of the keys
* - Collection<V> values() // Returns a collection view of the values
  + Set<Map.Entry<K,V>> entrySet() // Returns a set view of the mappings
* - K firstKey() // Returns the first (lowest) key
* - K lastKey() // Returns the last (highest) key
* - Map.Entry<K,V> firstEntry() // Returns a mapping of the first (lowest) key and its value
* - Map.Entry<K,V> lastEntry() // Returns a mapping of the last (highest) key and its value
  + Map.Entry<K,V> ceilingEntry(K key) // Returns the entry with the least key greater than or equal to the given key

**6.1 Additional**

* void putAll(Map<? extends K, ? extends V> m) // Copies all of the mappings from the specified map to this map
* V lowerKey(K key) // Returns the greatest key strictly less than the given key
* V floorKey(K key) // Returns the greatest key less than or equal to the given key
* V ceilingKey(K key) // Returns the least key greater than or equal to the given key
* V higherKey(K key) // Returns the least key strictly greater than the given key
* Map<K,V> subMap(K fromKey, K toKey) // Returns a view of the portion of this map whose keys range from fromKey, inclusive, to toKey, exclusive
* Map<K,V> headMap(K toKey) // Returns a view of the portion of this map whose keys are strictly less than toKey
* Map<K,V> tailMap(K fromKey) // Returns a view of the portion of this map whose keys are greater than or equal to fromKey