```java
package adt;

import java.util.Comparator;
import java.util.Iterator;

/**
 *
 * @author Chew Lip Sin
 * @author Lim Yi Leong
 * @param <T> type of elements stored in the stack.
 */
public interface ListInterface<T> {

    /**
     * Returns an iterator over the elements in the container.
     *
     * @param <T> type of elements stored in the List.
     * @return An iterator over the elements in the container.
     *
     */
    public Iterator<T> getIterator();

    /**
     * Adds the specified element to the end of the list.
     *
     * @param newEntry The element to add.
     * @return true if the addition is successful, or false if the list is full
     * Description: Adds a new entry to the end of the list. Entries currently
     * in the list are unaffected. The lists size is increased by 1.
     * Precondition: newEntry is not null. Post-condition:The entry has been
     * added to the list.
     *
     */
    public boolean add(T newEntry);

    /**
     * Adds the specified element to the list at the specified position.
     *
     * @param newPosition The position to add the element at.
     * @param newEntry The element to add.
     * @return true if the element was added successfully, false otherwise. *
     * Description: Adds a new entry at a specified position within the list.
     * Entries originally at and above the specified position are at the next
     * higher position within the list. The list size is increased by 1.
     * Precondition: newPosition >= 1 and newPosition smaller equal than
     * getLength()+1newEntry is not null. Post-condition:newEntry is added to
     * the list in the given position. The old entries have been shifted up one
     * position.
     */
    public boolean add(int newPosition, T newEntry);

    /**
     * Post-condition:The list is empty. Description:Removes all entries from
     * the list.
     */
    public void clear();

    /**
     * Checks whether the list contains the specified element.
     *
     * @param newEntry The element to check for.
     * @return true if the list contains the element, false otherwise.
     * Description: This method finds whether the new Entry exists or not.
     * Precondition: The array must exist. Post-condition:The array remains
     * unchanged
     */
    public boolean contains(T newEntry);

    /**
     * This method is used to retrieve the entry at a given position in the
     * list.
     *
     * @param givenPosition The position of the element to get.
     * @return a reference to the indicated entry or null, if either the list is
     * empty, givenPosition smaller 1, or givenPosition bigger getLength()
     * Precondition:The array must exist. Post-condition:The array remains
     * unchanged.
     */
    public T getEntry(int givenPosition);

    /**
     * Gets the number of entries currently in the list.
     *
     *
     * @return The number of entries currently in the list. Precondition:The
     * array must exist. Post-condition:The array remains unchanged.
     */
    public int size();

    /**
     * This method check if the array is empty
     *
     * @return true if the list is empty, false otherwise. * Post-condition:The
     * array remains unchanged.
     *
     */
    public boolean isEmpty();

    /**
     * Removes the element at the specified position in the list.
     *
     * @param givenPosition The position of the element to remove.
     * @return The element that was removed, or null if the position is invalid.
     */
    public T remove(int givenPosition);

    /**
     * Removes all occurrences of the specified elements from the list.
     *
     * @param elements The elements to be removed.
     * @return {@code true} if removal is successful, {@code false} if the list
     * is empty or elements are invalid.
     */
    public boolean removeAll(T... elements);

    /**
     * Adds all of the elements in the specified array to the end of the list.
     *
     * @param newElements The array of elements to add.
     * @return true if all of the elements were added successfully, false
     * otherwise. Precondition:newElements must not be null. Post-condition:
     */
    public boolean addAll(T... newElements);

    /**
     * Removes the first occurrence of the specified entry from the list.
     *
     * @param anEntry The entry to be removed.
     * @return {@code true} if removal is successful, {@code false} if the entry
     * is not found.
     */
    public boolean remove(T anEntry);

    /**
     * Replaces the entry at the specified position with the new entry.
     *
     * @param givenPosition The position of the entry to be replaced.
     * @param newEntry The new entry to replace the existing entry.
     * @return {@code true} if replacement is successful, {@code false} if the
     * list is empty, or position is invalid.
     */
    public boolean replace(int givenPosition, T newEntry);

    /**
     * Checks if the array is full.
     *
     * @return {@code true} if the array is full, {@code false} otherwise.
     */
    public boolean isFull();

    public <T extends Comparable<T>> void bubbleSort();

}
```