Project Description

Objective

Implement an extended ListADT that supports dynamic resizing and all standard list operations. Your list must store **Integer** objects and provide the following methods:

Original Methods:

1. **`boolean add(Integer e)`**
   Appends the specified integer to the end of the list. Returns `true` if the element is added successfully.

2. **`void add(int index, Integer element)`**
   Inserts the specified integer at the given index, shifting subsequent elements to the right.

3. **`boolean addAll(int index, Collection c)`**
   Inserts all elements from the provided collection (assumed to contain only integers) into the list starting at the specified index. Returns `true` if the list is modified.

4. **`boolean addAll(Collection c)`**
   Appends all elements from the provided collection to the end of the list. Returns `true` if the list is modified.

5. **`void clear()`**
   Removes all elements from the list.

6. **`boolean contains(Object o)`**
   Returns `true` if the list contains the specified object.

7. **`void ensureCapacity(int minCapacity)`**
   Increases the internal array's capacity (if needed) to ensure that it can hold at least the specified number of elements.

8. **`Integer get(int index)`**
   Returns the integer at the specified index.

New (Extended) Methods:

9. **`int indexOf(Object o)`**
   Returns the index of the first occurrence of the specified element, or `-1` if not found.

10. **`int lastIndexOf(Object o)`**
    Returns the index of the last occurrence of the specified element, or `-1` if not found.

11. **`boolean isEmpty()`**
    Returns `true` if the list is empty.

12. **`Integer remove(int index)`**
    Removes and returns the element at the specified index, shifting subsequent elements to the left.

13. **`boolean remove(Object o)`**
    Removes the first occurrence of the specified element from the list; returns `true` if the element was removed.

14. **`Integer set(int index, Integer element)`**
    Replaces the element at the specified index with the given element and returns the old element.

15. **`int size()`**
    Returns the number of elements currently in the list.

16. **`void trimToSize()`**
    Trims the capacity of the underlying array to be exactly the number of elements in the list.

17. **`String toString()`**
    Returns a string representation of the list.

Internal Array Management

- **Initial Capacity:**
  Start with an initial capacity (for example, 10).

- **Dynamic Resizing:**
  When adding elements, if the array is full, increase the capacity using the `ensureCapacity` method (e.g., double the current capacity or use the specified minimum capacity).

- **Trimming:**
  The `trimToSize` method should reduce the internal array size to exactly match the number of elements currently stored.

Testing Requirements

Your implementation must include a test program (or block) that manually verifies every method using if–else conditions. The tests should cover edge cases such as:

- Adding at the beginning, middle, and end.
- Inserting and removing elements when duplicates exist.
- Removing from an empty list.

- Verifying methods like `indexOf`, `lastIndexOf`, `isEmpty`, `set`, and `trimToSize`.
- Checking proper behavior after dynamic resizing.

Summary

- **Project Description:**
  You will implement an extended ListADT for Integer objects that supports dynamic array resizing along with standard list operations. Your implementation must include methods for adding, inserting, removing, searching, updating, and managing the capacity of the list.
- **Java Implementation:**
  The Java code provides an interface (optional) and a class (`ArrayListADT`) that implements all required methods. The `main` method contains tests (with if–else conditions) for every method and edge case.
- **Python Implementation:**
  The Python code implements the same functionality in a single class (`ArrayListADT`) and includes a test block under `if __name__ == "__main__":` to verify that every method works correctly.