Project Title: Generic ArrayList Implementation in Java

Overview

This project is an implementation of a dynamic, generic list data structure in Java that mimics the functionality of Java's built-in `ArrayList`. The implementation uses object-oriented programming (OOP) principles and Java generics to allow the list to store elements of any reference type. Additionally, comprehensive test cases are provided to ensure that the list works correctly with different data types such as `Integer`, `String`, and `Float`.

Project Components

1. **ListADT Interface**
    - **Description:**
      Defines a contract for a list data structure. This interface specifies all the essential operations that any list implementation must support, such as adding, removing, searching, and clearing elements.
    - **Key Methods:**
        - `boolean add(T e)`
        - `void add(int index, T element)`
        - `boolean addAll(int index, Collection<? extends T> c)`
        - `boolean addAll(Collection<? extends T> c)`
        - `void clear()`
        - `boolean contains(Object o)`
        - `void ensureCapacity(int minCapacity)`
        - `T get(int index)`
        - `int indexOf(Object o)`
        - `int lastIndexOf(Object o)`
        - `boolean isEmpty()`
        - `T remove(int index)`
        - `boolean remove(Object o)`
        - `T set(int index, T element)`
        - `int size()`
        - `void trimToSize()`

- `String toString()`

2. **ArrayListADT Class**

   - **Description:**
     Implements the `ListADT` interface using a dynamically resizable array as its underlying data store. This class is generic, meaning it can store elements of any type specified at instantiation.
   - **Key Features:**
     - **Generic Array Creation:**
       Uses Java reflection (`Array.newInstance`) to create an array of type `T[]` safely. This avoids runtime issues like `ClassCastException` by ensuring the array is of the correct type.
     - **Dynamic Resizing:**
       Automatically resizes the internal array when the number of elements exceeds the current capacity.
     - **Core Operations:**
       Implements all list operations including adding, inserting, removing, and searching elements. The methods ensure that the list maintains its order and integrity after each operation.
   - **Encapsulation:**
     The internal data array and the size counter are encapsulated within the class. Only methods defined in the interface are exposed to the client code.

3. **ListADTTestGenerics Class**

   - **Description:**
     A test harness that verifies the correctness of the `ArrayListADT` implementation. It contains separate test methods for `Integer` , `String` , and `Float` data types.
   - **Test Cases:**
     - **Adding Elements:**
       Tests the `add(T)` and `add(int, T)` methods.
     - **Inserting Collections:**
       Verifies that `addAll(int, Collection<? extends T>)` and `addAll(Collection<? extends T>)` work as expected.
     - **Searching and Indexing:**
       Checks `contains(Object)` , `indexOf(Object)` , and `lastIndexOf(Object)` methods.
     - **Removal and Replacement:**
       Ensures that `remove(int)` , `remove(Object)` , and `set(int, T)` operate correctly.
     - **Capacity Management:**

Confirms that `ensureCapacity(int)` and `trimToSize()` adjust the internal storage as needed.

- **Clearing the List:**
  Validates the `clear()` method to make sure all elements can be removed.