Problem Statement

Objective

Design and implement a `ListADT` in Java that uses an underlying array to store **Integer** objects. The implementation must not use Java's built-in collection classes (such as ArrayList) or generics. Instead, you will explicitly work with `Integer` objects. The list should provide basic operations for adding, retrieving, and clearing elements, as well as for checking whether an element exists in the list.

Abstract Data Type (ADT) Definition

An Abstract Data Type (ADT) defines a data structure purely in terms of the operations that can be performed on it and the behavior that the operations guarantee, without specifying the details of its implementation. In this exercise, the `ListADT` is defined by the following operations:

- **add(Integer e):**
  Appends the specified `Integer` to the end of the list.
  Behavior: Returns `true` if the integer is successfully added.

- **add(int index, Integer element):**
  Inserts the specified `Integer` at the given position in the list.
  Behavior: The element is placed at the specified index, shifting any subsequent elements to the right.

- **addAll(int index, Collection c):**
  Inserts all of the `Integer` elements from the provided collection into the list, starting at the specified index.
  Behavior: Returns `true` if the list changes as a result of this operation.

- **clear():**
  Removes all elements from the list.
  Behavior: The list is left empty after this operation.

- **contains(Object o):**
  Checks if the list contains the specified object.
  Behavior: Returns `true` if the list contains the object; otherwise, returns `false` .

- **ensureCapacity(int minCapacity):**
  Increases the internal capacity of the list, if necessary, so that it can hold at least the number of elements specified by `minCapacity` .
  Behavior: If the current array is not large enough, the method expands it.

- **get(int index):**

Returns the `Integer` at the specified index.
Behavior: Retrieves the element at the given position in the list.

- **toString():**
  Returns a string representation of the list.
  Behavior: The string shows the list's elements in order.

Implementation Requirements

1. **Interface Definition:**
   Create an interface `ListADT` that declares the methods described above. (Note: The method `boolean addAll(Collection c)` has been removed.) Use the following method signatures:

   - `boolean add(Integer e)`
   - `void add(int index, Integer element)`
   - `boolean addAll(int index, Collection c)`
   - `void clear()`
   - `boolean contains(Object o)`
   - `void ensureCapacity(int minCapacity)`
   - `Integer get(int index)`
   - `String toString()`

2. **Class Implementation:**
   Create a class (for example, `ArrayListADT`) that implements the `ListADT` interface. This class should:

   - Use an array of `Integer` as its internal data structure.
   - Start with an initial capacity (for example, 10).
   - Dynamically resize the internal array when more capacity is required (using the `ensureCapacity` method).
   - Assume that all indices provided (for methods like `get` and `add(int, Integer)`) are valid (no exception handling for invalid indices is required).

3. **Testing:**
   Write a test program with a `main` method to demonstrate that your implementation works as expected. The test program should verify that each of the operations behaves according to the ADT definition.

Sample Interface Definition

Below is an example of what your `ListADT` interface might look like:

```java
import java.util.Collection;

public interface ListADT {
    /**
     * Appends the specified Integer to the end of this list.
     * @param e the Integer to be appended
     * @return true if the Integer was added successfully, false otherwise.
     */
    boolean add(Integer e);

    /**
     * Inserts the specified Integer at the specified position in this list.
     * @param index the index at which the element is to be inserted
     * @param element the Integer to be inserted
     */
    void add(int index, Integer element);

    /**
     * Inserts all of the elements in the specified collection (assumed to contain I
     * into this list, starting at the specified position.
     * @param index the index at which to start inserting the first element from the
     * @param c the collection containing Integer elements to be added to this list
     * @return true if the list changed as a result of the call
     */
    boolean addAll(int index, Collection c);

    /**
     * Removes all of the elements from this list.
     */
    void clear();

    /**
     * Returns true if this list contains the specified element.
     * @param o the element whose presence in this list is to be tested
     * @return true if this list contains the specified element, false otherwise
     */
    boolean contains(Object o);

    /**
     * Increases the capacity of this list instance, if necessary, to ensure that it
     * at least the number of elements specified by the minCapacity argument.
     * @param minCapacity the desired minimum capacity
     */
    void ensureCapacity(int minCapacity);

    /**
     * Returns the Integer at the specified position in this list.
     * @param index the index of the element to return
     * @return the Integer at the specified position in this list
     */
    Integer get(int index);

    /**
     * Returns a string representation of this list.
     * @return a string representation of the list
     */
    String toString();
}
```

Guidelines for Your Implementation

- **Internal Array:**
  Begin with an initial array size (e.g., 10) for storing `Integer` objects. When the number of elements exceeds the current capacity, use the `ensureCapacity` method to expand the array.
- **Adding Elements:**
  For methods `add(Integer)` and `add(int, Integer)` , check whether there is enough space in the internal array before adding a new element. If the array is full, call `ensureCapacity` to resize it.
- **Collection Method:**
  In the `addAll(int, Collection)` method, iterate through the provided collection (assume it contains only `Integer` objects) and add each element to your list starting at the specified index.
- **Testing:**
  Create a test class with a `main` method that demonstrates each method's functionality. Make sure the list's behavior conforms to the ADT definition provided above.