ArrayListADT - step by step (no code)

Let's build the `ArrayListADT` class from scratch, step-by-step, focusing on completing each function required by the `AbstractList<E>` interface. We'll use an array as the underlying storage and explain everything along the way, keeping it beginner-friendly since you're new to arrays.
Let's build the `ArrayListADT` class from scratch, step-by-step, focusing on completing each function required by the `AbstractList<E>` interface.

Step 1: Set Up the Class Foundation

We need a class that uses an array to store elements and a variable to track the number of items. Let's start with the basic structure.

- **What's Happening:**
  - `E[] elements`: An array to store items of type `E` (e.g., `Integer`, `String`). We create it as `Object[]` and cast it because Java requires this for generics.
  - `nextIndex`: Tracks how many items are in the array and where to add the next one.
  - Constructor: Makes an array with 5 slots, all empty (null), and sets `nextIndex` to 0.

Step 2: Implement Each Method from Scratch

We'll go through each method in the `AbstractList<E>` interface, writing them one by one.

1. `boolean add(E element)`

This adds an item to the end of the list. If the array is full, we'll make a bigger one.

- **Step-by-Step:**
  - Check if `nextIndex` equals `elements.length` (e.g., 5 items in 5 slots = full).
  - If full, create a new array with double the size (e.g., 10 slots).
  - Copy all items from `elements` to `newArray`.
  - Replace `elements` with `newArray`.
  - Put the new `element` at `nextIndex` and increase `nextIndex`.
- **Example:**
  - Start: `[null, null, null, null, null]`, `nextIndex = 0`.
  - `add(10)`: `[10, null, null, null, null]`, `nextIndex = 1`.
  - After 5 adds: `[10, 20, 30, 40, 50]`, `nextIndex = 5`.
  - `add(60)`: New array `[10, 20, 30, 40, 50, 60, null, ...]`, `nextIndex = 6`.

---

2. `boolean add(int index, E element)`

This inserts an item at a specific position, shifting others right.

- **Step-by-Step:**
  - Check `index`: It can be 0 up to `nextIndex` (to add at the end).
  - If the array is full, double its size like in `add(E element)`.
  - Shift items right: Move each item one slot forward from `index` to the end.
  - Place `element` at `index`.
  - Increase `nextIndex`.
- **Example:**
  - `[10, 20, 30, null, null]`, `nextIndex = 3`.
  - `add(1, 15)`:

- Shift: `[10, 20, 30, null, null]` → `[10, 20, 30, 30, null]`.
- Shift: `[10, 20, 20, 30, null]`.
- Insert: `[10, 15, 20, 30, null]`, `nextIndex = 4`.

---

3. `E get(int index)`

This gets the item at a specific position.

- **Step-by-Step:**
  - Validate `index`: Must be between 0 and `nextIndex - 1`.
  - Return the item from the array at that index.
- **Example:**
  - `[10, 20, 30, null, null]`, `nextIndex = 3`.
  - `get(1)` returns `20`.
  - `get(3)` throws an exception.

---

4. `E set(int index, E element)`

This replaces an item at a position and returns the old one.

- **Step-by-Step:**
  - Validate `index`.
  - Store the current item at `index`.
  - Replace it with the new `element`.
  - Return the old item.
- **Example:**
  - `[10, 20, 30]`, `nextIndex = 3`.
  - `set(1, 25)`: Returns `20`, array becomes `[10, 25, 30]`.

---

5. `E remove(int index)`

This removes an item and shifts others left.

- **Step-by-Step:**
  - Validate `index`.
  - Save the item at `index`.
  - Shift all items after `index` one spot left.
  - Set the last spot to `null` (it's now empty).
  - Decrease `nextIndex`.
- **Example:**
  - `[10, 20, 30, 40]`, `nextIndex = 4`.
  - `remove(1)`:
    - Save `20`.
    - Shift: `[10, 30, 30, 40]`, then `[10, 30, 40, 40]`.
    - Clear: `[10, 30, 40, null]`, `nextIndex = 3`.
    - Return `20`.

---

6. `int size()`

This tells us how many items are in the list.

- **Step-by-Step:**

- Just return `nextIndex`, which tracks the number of items.

- **Example:**
  - `[10, 20, 30, null, null]`, `nextIndex = 3`.
  - `size()` returns `3`.

---

7. `int indexOf(E element)`

This finds the first position of an item.

- **Step-by-Step:**
  - Loop through the array up to `nextIndex`.
  - Compare `element` with each item:
    - If `element` is `null`, check for `null`.
    - Otherwise, use `equals()` to compare objects.
  - Return the index if found, or `-1` if not.
- **Example:**
  - `[10, 20, 30]`, `nextIndex = 3`.
  - `indexOf(20)` returns `1`.
  - `indexOf(40)` returns `-1`.

---

8. `boolean contains(E element)`

This checks if an item is in the list.

- **Step-by-Step:**
  - Call `indexOf`. If it returns `-1`, the item isn't there (`false`).
  - Otherwise, it's found (`true`).
- **Example:**
  - `[10, 20, 30]`, `nextIndex = 3`.
  - `contains(20)` returns `true`.
  - `contains(40)` returns `false`.

---

9. `boolean isEmpty()`

This checks if the list has no items.

- **Step-by-Step:**
  - Check if `nextIndex` is `0` (no items added).
- **Example:**
  - `[null, null, null, null, null]`, `nextIndex = 0` → `true`.
  - `[10, 20, null, null, null]`, `nextIndex = 2` → `false`.

Step 3: Complete the Code

Complete full `ArrayListADT` with all methods

Step 4: Test It

Let's test it to make sure everything works:

```
class ArrayListADTRunner {
    public static void main(String[] args) {
        ArrayListADT<Integer> list = new ArrayListADT<>();
        list.add(10);
        list.add(20);
        list.add(30);
        System.out.println("Size: " + list.size()); // 3
        System.out.println("Get 1: " + list.get(1)); // 20
        list.add(1, 15);
        System.out.println("Get 1: " + list.get(1)); // 15
        System.out.println("Contains 20: " + list.contains(20)); // true
        list.remove(0);
        System.out.println("Get 0: " + list.get(0)); // 15
        System.out.println("Size: " + list.size()); // 3
        System.out.println("Is empty: " + list.isEmpty()); // false
    }
}
```