# Why Arrays Can't Grow, and Lists Can

*Collecting failing test IDs during a run means you don't know the count in advance -- the exact situation an array can't handle, and a list was built for.*

Automation Foundations · ashishautomationlabs.store

Suppose you are collecting the IDs of failing tests during a run. You do not know in advance how many will fail -- that is the entire point of running the tests. With an array you are stuck before you start, because you must state the size up front.

Guess too small and you overflow: the eleventh failure has nowhere to go, and you get an out-of-bounds crash. Guess too large and you waste space, and worse, you leave misleading defaults behind -- an array of 100 built to hold 3 failures also holds 97 boxes of `null` that later code must remember to ignore. And an array has no `add` method at all. To truly grow one, you would create a bigger array and copy every element across by hand, each time.

This is not a flaw in arrays. A fixed row of boxes is exactly right when the count is known and stable -- the twelve months of a year, the eight primitive types, a chessboard. But for test data that grows as it goes, you need a container built to grow. Java calls it a **list**.

## ArrayList: the container that grows

An `ArrayList` is a list that starts empty and grows as you add to it. It lives in Java's standard library, so you bring it in with two import lines. Then you create one and add to it with `.add(...)`, and it stretches to fit -- no size declared, ever:

```
import java.util.ArrayList;
import java.util.List;

public class ListDemo {
    public static void main(String[] args) {
        List<String> failures = new ArrayList<>();
        String[] results = {"PASS", "FAIL", "PASS", "FAIL", "FAIL"};
        for (int i = 0; i < results.length; i++) {
            if (results[i].equals("FAIL")) {
                failures.add("TC-" + (1000 + i));
            }
        }
        System.out.println("Failures found: " + failures.size());
        for (String id : failures) {
            System.out.println("  " + id);
        }
    }
```

```
}

Failures found: 3
  TC-1001
  TC-1003
  TC-1004
```

That is the pattern at the heart of real test reporting: start with an empty list, walk your results, and add each failure as you find it. The list was never given a size. It held zero, then one, then two, then three, growing on demand.

## The one new syntax piece

Read the declaration slowly, because it has one new part: `List<String> failures = new ArrayList<>();`. The `<String>` says "a list of Strings" -- it tells Java what type of thing lives inside, so it can stop you adding the wrong type by mistake. The angle brackets are called **generics**, and for now that one-line understanding is all you need; you'll use them long before you need their full theory.

> *An array's size is locked the moment you create it. A list starts at zero and grows with every* `add` *-- it carries its own size, so it is always exactly as long as the number of items you put in.*