# Why List on the Left Finally Makes Sense

*List<String> x = new ArrayList<>() is the exact same idea as the famous WebDriver line, copied on faith since Chapter 5 and now fully understood.*

Automation Foundations · ashishautomationlabs.store

Look again at a line you have used for seven chapters:

```
List<String> failures = new ArrayList<>();
```

Chapter 5 said this was good style and asked you to copy the pattern. Now you can read it. `List` is an interface. `ArrayList` is one class that implements it, and `LinkedList` is another. You declare the general capability and create one specific kind:

```
List<String> a = new ArrayList<>();
List<String> b = new LinkedList<>();
fill(a);
fill(b);

ArrayList  : [LoginTest, CartTest]
LinkedList : [LoginTest, CartTest]
Same type? : false
```

One `fill` method took both, because both are a `List`. The objects are genuinely different types -- the last line proves it -- yet every line of code that uses them is identical. This is called **programming to an interface**, and it is why the pattern was worth adopting before you understood it. The same idea explains why Chapter 10 had you write `Map<String, String>` on the left of a `new HashMap<>()` rather than `HashMap<String, String>` -- one capability, swappable implementation, every time.

There is more to an interface than a list of empty promises, though. Modern Java lets one carry an actual body -- and that changes the question of when to reach for an interface at all versus something with more structure.