Packages: Folders With Meaning

A package becomes part of a class's real name, and grouping classes by what they do -- tests, pages, utils, models -- is what lets a newcomer guess where anything lives.

Automation Foundations · ashishautomationlabs.store

Your framework can now bend at the joints. It still cannot find anything. Picture the folder as it stands: twelve test classes, a handful of page objects, a `BaseTest`, two utility classes, and a file of test data. All of it sits in one directory, every class marked public, with the string `"https://uat.shopcart.internal"` typed out in nine different places. It works. It runs. And the first colleague who opens it will need twenty minutes to find where a URL comes from.

A **package** is a named group of classes. It matches a folder on disk, and it becomes part of the class's real name:

```
package com.shopcart.utils;
public class UrlBuilder {
    public static String forEnv(String env) {
        return "https://" + env + ".shopcart.internal";
    }
}
```

```
Package  : com.shopcart.utils
Full name: com.shopcart.utils.UrlBuilder
Built    : https://uat.shopcart.internal
```

The package line must be the first statement in the file, and the folder structure must match it: `com/shopcart/utils/UrlBuilder.java`. That looks like bureaucracy until you notice what the last two output lines say. The class's real name is `com.shopcart.utils.UrlBuilder`, not `UrlBuilder`. The package is part of its identity. That solves a genuine problem: two teams can both write a `Config` class, and as long as they sit in different packages, Java keeps them apart. It is also why every library you use has a long name: `java.util.List`, `org.openqa.selenium.WebDriver`. The reverse-domain convention -- `com.yourcompany.project` -- exists so two organizations never collide.

For a test framework, the usual layout follows what the classes do:

- `com.shopcart.tests` -- `LoginTest`, `CartTest`
- `com.shopcart.pages` -- `LoginPage`, `BasePage`
- `com.shopcart.utils` -- `UrlUtil`, `TestConstants`
- `com.shopcart.models` -- `TestResult`, `Defect`

One package, one job -- a layout a newcomer can guess. Four or five packages is typical for a framework; split a package that needs a paragraph to explain it, and merge if you have twenty packages holding two classes each.

You have been using `import` since Chapter 5 without comment. Now it is readable: `import java.util.List;` tells the compiler which `List` you mean, so you can write the short name in your code. Classes in the same package need no import at all -- which is the first hint about access levels, because "same package" is a real boundary in Java.

That boundary is worth naming precisely, because there are four of them in total, and two of the four are the ones most people never learn properly.