One Script, Every Browser

A loop that calls methods on a Browser never mentions Chrome, Firefox, or Edge by name -- because the method that runs is chosen from the actual object at the moment of the call, not the declared type.

Automation Foundations · ashishautomationlabs.store

Here is why any of this matters. The `Browser` contract is signed by three classes, and the test code below knows about none of them:

```
String[] wanted = {"chrome", "firefox", "edge"};
for (String name : wanted) {
    Browser b = create(name);
    b.open("https://uat.shopcart.internal/login");
    b.close();
}
```

```
[Chrome ] navigating to https://uat.shopcart.internal/login
[Chrome ] closed
[Firefox] navigating to https://uat.shopcart.internal/login
[Firefox] closed
[Edge   ] navigating to https://uat.shopcart.internal/login
[Edge   ] closed
```

Look at what the loop body does not contain. No `if` for the browser type. No mention of Chrome, Firefox, or Edge. It calls `open` and `close` on something that is a `Browser`, and the right code runs every time.

That is **runtime polymorphism**: the method is chosen from the actual object at the moment of the call, not from the declared type at compile time. It is the same mechanism you saw with `BaseTest t = new ApiTest()` in Chapter 11, now doing the job it was built for. This is precisely how cross-browser automation works, and it is the answer to "why is `WebDriver` an interface?" Because a test written against the interface is written against every browser at once. Add a fourth browser tomorrow, and the loop above does not change.

The declared type and the actual object are genuinely different things, and you can see both:

```
Browser b = new FirefoxBrowser();
System.out.println("Declared type : Browser");
System.out.println("Actual object : " + b.getClass().getSimpleName());
```

```
[Firefox] navigating to https://uat.shopcart.internal
Declared type : Browser
Actual object : FirefoxBrowser
```

Storing a `FirefoxBrowser` in a `Browser` variable is called **upcasting**, and Java does it automatically because a `FirefoxBrowser` genuinely is a `Browser`. The variable limits what you may

call -- only what the contract lists -- while the object decides what actually runs.

> *Given* `Browser b = new FirefoxBrowser();` *-- which class's* `open` *method runs? Can you call a method that exists only on* `FirefoxBrowser` *and not on* `Browser`*? Why can you not write* `new Browser()`*?*

>

> *Answer --* `FirefoxBrowser`*'s, because the object decides. No -- the declared type limits what you may call. And an interface has no method bodies, so there is nothing to run.*

This exact pattern -- declare the general capability, create one specific kind -- is not new to this chapter. You've been writing it since Chapter 5, on faith.