An Interface Is a Contract

An interface lists method signatures with no bodies -- the promise without the implementation. Sign the contract and skip a method, and the compiler refuses to build.

Automation Foundations · ashishautomationlabs.store

This is the line. If you have watched a single automation tutorial, read one framework, or sat one interview, you have met it:

```
WebDriver driver = new ChromeDriver();
```

It is the most-asked Java question in test automation, and every other piece of it is now familiar. `new ChromeDriver()` builds an object, from Chapter 7. `driver` is a reference variable holding an arrow to it, from Chapter 2. Chapter 11 showed you that the declared type and the actual object can differ, and that Java picks methods from the object. The one word still unexplained is `WebDriver` itself. It is not a class. You cannot write `new WebDriver()`. It is an **interface**.

A class says what something is and how it works. An interface says only what something must be able to do. It lists method signatures with no bodies -- the promises, without the implementation:

```
interface Browser {
    void open(String url);
    void close();
}
```

Read that as a contract. Any class that signs it must supply real versions of `open` and `close`. The interface itself supplies nothing and cannot be instantiated: `new Browser()` is meaningless, because there is no code behind those methods. A class signs the contract with `implements`:

```
class ChromeBrowser implements Browser {
    @Override
    public void open(String url) {
        System.out.println("[Chrome ] navigating to " + url);
    }
    @Override
    public void close() {
        System.out.println("[Chrome ] closed");
    }
}

Browser b = new ChromeBrowser();
b.open("https://uat.shopcart.internal");
b.close();
```

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

Two details interviewers probe. The compiler enforces the contract: leave out `close` and `ChromeBrowser` will not compile, because it claimed to be a `Browser` and is not. And interface methods are public -- you cannot narrow them when you implement, since the whole point is that anyone may call them.

Note the declaration in the calling code: `Browser b = new ChromeBrowser();`. That is the same shape as the famous line. You have already written it.

One more thing worth settling now: why does Java allow a class to implement any number of interfaces, when Chapter 11 limited it to a single superclass? Because interfaces historically carried no implementation, so there was nothing to conflict. Two parent classes could each supply a different version of the same method, and Java refuses to guess which one you meant. Interfaces let a class make several promises without inheriting several bodies.

`ChromeBrowser` signing the contract is one class doing one job. The real point of an interface only shows up once there's more than one class signing it -- and that's where this gets its payoff.