extends: One Class Built on Another

class LoginTest extends BaseTest -- the first line of nearly every test class in the world. A subclass receives the fields and methods of its parent and may use them as its own.

Automation Foundations · ashishautomationlabs.store

Open any Java test framework in the world and look at the first line of a test class. It will look like this:

```
public class LoginTest extends BaseTest {
```

That word `extends` is doing a great deal of work. It is the reason a test class can open a browser without containing a single line of browser code. It is why your team's framework has one setup method instead of two hundred. You already know what it is for, from your own working life: when your team writes test cases, there is a standard preamble that every case inherits -- the environment, the login steps, the test data setup -- written once and referenced everywhere. Nobody retypes it into all two hundred cases. `extends` is Java's version of that arrangement, with one improvement: the compiler enforces it.

Here is the smallest real example. A `BaseTest` holds the setup and teardown that every test needs. A `LoginTest` extends it and adds only its own check:

```
class BaseTest {
    String environment = "UAT-2";
    void setUp() {
        System.out.println("[BASE] Opening browser on " + environment);
    }
    void tearDown() {
        System.out.println("[BASE] Closing browser");
    }
}
class LoginTest extends BaseTest {
    void runCheck() {
        System.out.println("[TEST] Logging in on " + environment);
    }
}

LoginTest t = new LoginTest();
t.setUp();
t.runCheck();
t.tearDown();

[BASE] Opening browser on UAT-2
[TEST] Logging in on UAT-2
[BASE] Closing browser
```

Read what happened. `LoginTest` declares exactly one method, `runCheck`. Yet the object answered `setUp()` and `tearDown()` as though it owned them, and `runCheck` used the field `environment` that it never declared. That is inheritance: a subclass receives the fields and methods of its parent, and may use them as its own.

The vocabulary is worth fixing now, because interviewers use all of it. `BaseTest` is the **superclass**, also called the parent or base class. `LoginTest` is the **subclass**, also called the child or derived class. And the relationship `extends` creates is called **is a**: a `LoginTest` is a `BaseTest`. Remember that phrase -- it is the test for whether inheritance is the right tool at all, and this chapter returns to it at the end.

Java allows one superclass only -- a class may not extend two parents. That restriction has a reason, and the tool that works around it is the interface, which is the next chapter. When you need a class to satisfy several contracts instead of inheriting from several parents, that's what interfaces are for; a class may implement as many of those as it likes.

A subclass inherits fields and methods, but not everything. One thing is deliberately excluded, and reaching around that exclusion is the whole reason the keyword `super` exists.