Overriding Versus Overloading

These two words look alike and mean different things. Change a parameter type even slightly and you haven't overridden a method -- you've silently added a second one, and the original still answers every call.

Automation Foundations · ashishautomationlabs.store

These two words look alike and mean different things. This is the most commonly confused pair in this chapter, and a reliable interview question.

Overriding replaces a parent method: same name, same parameter list, in a subclass. **Overloading** is what you met in Chapter 8: same name, different parameter list, in the same class or anywhere else. Overriding is about inheritance; overloading is not.

Confuse them and you get a silent defect. Here a subclass means to override `logStep`, but writes a `StringBuilder` parameter instead of a `String`:

```
class BaseTest {
    void logStep(String step) {
        System.out.println("[BASE] " + step);
    }
}
class SmokeTest extends BaseTest {
    void logStep(StringBuilder step) {
        System.out.println("[SMOKE] " + step);
    }
}

BaseTest t = new SmokeTest();
t.logStep("open the cart");

[BASE] open the cart
```

The subclass version never ran. Because the parameter type differs, `SmokeTest` did not override anything -- it added a second, unrelated method, and the original was still there to answer the call. No error, no warning, and a subclass whose custom logging silently does nothing.

Add `@Override` to that method and the program does not compile at all. The compiler says, in effect, "you claimed to override something, and there is nothing here to override." That is exactly the report you want, at exactly the right time.

One more trap wears the same disguise: can you override a static method? No. Overriding works on objects, and a static method belongs to the class rather than to any object, as you learned in Chapter 8. Declaring a static method with the same signature in a subclass _hides_ the parent's version rather than overriding it, and which one runs depends on the declared type of the variable

-- the opposite of everything you just learned about overriding. Avoid it.

Overriding and overloading are both about methods. There is a third relationship in this chapter that works completely differently, because it applies to access, not behavior -- and it exists specifically because a base class needs it.