super, and the Parent That Goes First

Constructors aren't inherited, so a subclass constructor must arrange for the parent's constructor to run first. That's super -- and interviewers ask about the order for a reason.

Automation Foundations · ashishautomationlabs.store

Two situations need you to name the parent explicitly.

The first is constructors. Constructors are not inherited, so when your subclass has a constructor, it must arrange for the parent's constructor to run first. That is `super(...)`:

```
class BaseTest {
    String name;
    BaseTest(String name) {
        this.name = name;
        System.out.println("1. BaseTest constructor: " + name);
    }
}
class LoginTest extends BaseTest {
    LoginTest(String name) {
        super(name);
        System.out.println("2. LoginTest constructor: " + name);
    }
}
```

```
1. BaseTest constructor: LoginTest
2. LoginTest constructor: LoginTest
```

Note the order, because interviewers ask about it: the parent's constructor finishes before the child's body runs. That is the only sensible order -- the child may depend on fields the parent sets up, so the parent must go first. `super(...)` must be the very first statement in the constructor, and if you leave it out entirely, Java inserts a call to the parent's no-argument constructor for you. When the parent has no such constructor, the compiler stops you -- and now you know why.

The second use is calling a parent method you have replaced. Inside an overridden method, `super.setUp()` runs the parent's version -- which is how a subclass adds to inherited behavior rather than discarding it.

The parent going first solves a real ordering problem, but it doesn't yet let you change what a method actually does once you have it. That's the next, and more interesting, use of the parent-child relationship.