Constructors, this, and Your First Method

A constructor builds an object fully formed at new. this is how you tell a field apart from a parameter that shares its name -- get it wrong, and the field stays empty.

Automation Foundations · ashishautomationlabs.store

Setting every field by hand after `new` is tedious and easy to forget -- miss one line and a field is left empty. A **constructor** fixes that. It's a special block that runs when an object is created, so the object arrives fully built. While you're adding behavior, give the class a method too -- an operation that belongs to the object:

```
class TestResult {
    String name;
    String status;
    int durationMs;

    TestResult(String name, String status, int durationMs) {
        this.name = name;
        this.status = status;
        this.durationMs = durationMs;
    }

    boolean isPassed() {
        return status.equals("PASS");
    }
}

TestResult login  = new TestResult("LoginTest", "PASS", 240);
TestResult search = new TestResult("SearchTest", "FAIL", 512);
System.out.println(login.name + " passed? " + login.isPassed());
System.out.println(search.name + " passed? " + search.isPassed());
```

```
LoginTest passed? true
SearchTest passed? false
```

this -- one keyword, one reason it exists

The constructor has the same name as the class and no return type, and it runs once, at `new`. Inside it you see `this`, the keyword that means _this particular object_. So `this.name = name` reads as: set this object's `name` field to the `name` that was passed in. The reason `this` is needed here is that the parameter and the field share the name `name` -- `this.name` says which one you mean. Get this wrong and you get a real bug, one worth its own article later in this chapter.

`isPassed()` is a **method** -- behavior that belongs to the object. It uses the object's own `status` field, and notice it compares with `.equals`, exactly as Chapter 6 taught. A `TestResult` now carries its data and knows how to answer a question about itself.