COMP 3550

4.2 — TEST DOUBLES INTRO: DUMMIES, STUBS, FAKES, MOCKS

Week 4: Exceptional Testing & Technical Debt

WHY USE A TEST DOUBLE?

Real code talks to real systems — and that can cause real trouble in tests.

- 🐢 Slow: Databases, APIs, and file systems
- 🧨 Risky: Might send emails, delete data, make network calls
- 💸 Expensive: CI pipelines get bogged down

WHY USE A TEST DOUBLE?

Real code talks to real systems — and that can cause real trouble in tests.

- 🐢 Slow: Databases, APIs, and file systems
- 🧨 Risky: Might send emails, delete data, make network calls
- 💸 Expensive: CI pipelines get bogged down

The Solution: Test Doubles

- *Fake collaborators you control for testing*
- Replace real dependencies with safe, predictable ones
- Let you test behavior without real side effects
- Make tests fast, isolated, and reliable

WHY USE A TEST DOUBLE?

Real code talks to real systems — and that can cause real trouble in tests.

- 🐢 Slow: Databases, APIs, and file systems
- 🧨 Risky: Might send emails, delete data, make network calls
- 💸 Expensive: CI pipelines get bogged down

The Solution: Test Doubles

- *Fake collaborators you control for testing*
- Replace real dependencies with safe, predictable ones
- Let you test behavior without real side effects
- Make tests fast, isolated, and reliable

"I don't need a real jet engine to test the flight controls in a simulator."

This is **just** an intro to test doubles, we will use a simplified database for now and learn more advanced techniques by iteration 3.

TYPES OF TEST DOUBLES

Type	What It Does	Example Use
Dummy	Passed but never used	new User("", "", "") just to satisfy method signature
Stub	Returns a fixed value	Return "admin" for getRole()
Fake	Works, but simplified logic	In-memory DB using ArrayList
Mock	Verifies if something was called	Was sendEmail() triggered?
Spy	Records actual calls + data used	Did save() get called with this object?

TYPES OF TEST DOUBLES: SUMMARY

- **Dummy**: Needed for the method, but irrelevant
- **Stub**: Answers questions
- **Fake**: Actually does something, just simpler
- **Mock**: Confirms something happened
- **Spy**: Confirms what happened and with what

In your iteration 1 you will be starting with Fakes/Stubs (Occasionally) as these are easiest without a framework (Iteration 3)

SWAPPING IN TEST DOUBLES

[Test] → Service → Database

- Test calls Service which depends on Database
- Not ideal for testing (slow, risky, hard to control)

SWAPPING IN TEST DOUBLES

[Test] → Service → Database

- Test calls Service which depends on Database
- Not ideal for testing (slow, risky, hard to control)

[Test] → Service → FakeyDatabase (ArrayList-based)

- FakeDAO replaces real DAO
- No real DB involved
- Fast, predictable, safe

I1 will ONLY have Fakey Database classes. We will move on to proper swapping in I2.

You're not testing the DB — you're testing how the service behaves when the DAO gives it certain data.

HOW WILL WE DO THE SWAPPING THOUGH?

INTERFACES

HOW WILL WE DO THE SWAPPING THOUGH?

```
public class RealEmailSender implements EmailSender {
    public void send(String to, String subject, String body) {
        // Connect to SMTP server
        // Actually send the email
    }
}
```

Not safe in tests!

- Might spam people
- Needs config
- Slow & flaky

HOW WILL WE DO THE SWAPPING THOUGH?

```
public class RealEmailSender implements EmailSender {
    public void send(String to, String subject, String body) {
        // Connect to SMTP server
        // Actually send the email
    }
}
```

Not safe in tests!

- Might spam people
- Needs config
- Slow & flaky

HOW WILL WE DO THE SWAPPING THOUGH?

```java
public class StubEmailSender implements EmailSender {
    private List<Email> sentEmails = new ArrayList<>();

    public void send(String to, String subject, String body) {
        sentEmails.add(new Email(to, subject, body));
        System.out.println("Stub: Email to " + to);
    }

    public List<Email> getSentEmails() {
        return sentEmails;
    }
}
```

Safe and testable!

- Logs or stores messages
- Lets you verify what would've been sent
- No actual side effects

Some of your database classes in iteration 1 might look like this (stubby) and some may have a bit more to them (fakey). Both are okay for iteration 1!

INTRO TO MOCKITO (MORE COMING IN ITERATION 3: ADVANCED TESTING)

- Lets you create mock objects to control behavior and verify interactions.

Why It's Powerful:

- No need to write your own stubs/fakes
- Precise control over behavior
- Clear verification of interactions

```
// 1. Create a mock
EmailSender mockSender = mock(EmailSender.class);

// 2. Stub behavior
when(mockSender.send(any(), any(), any())).thenReturn(true);

// 3. Use in code under test
service.registerUser(...);

// 4. Verify interaction
verify(mockSender).send("bob@example.com", "Welcome", "...");
```

INTRO TO MOCKITO (MORE COMING IN ITERATION 3: ADVANCED TESTING)

- Lets you create mock objects to control behavior and verify interactions.

Why It's Powerful:

- No need to write your own stubs/fakes
- Precise control over behavior
- Clear verification of interactions

```
// 1. Create a mock
EmailSender mockSender = mock(EmailSender.class);

// 2. Stub behavior
when(mockSender.send(any(), any(), any())).thenReturn(true);

// 3. Use in code under test
service.registerUser(...);

// 4. Verify interaction
verify(mockSender).send("bob@example.com", "Welcome", "...");
```

Better Options (Sometimes):

- Use fakes when you want real-ish behavior
- Use stubs when you only care about return values
- Use mocks when interaction is the whole point

Rule of Thumb (in the real world):
Don’t mock what you don’t own.
Use mocks to test interactions, not outcomes.

BEST USE CASES FOR TEST DOUBLES

🐢 1. Slow or External Services
- Databases, web APIs, 3rd-party services
- Test doubles make tests fast and local

🧨 2. Services with Side Effects
- Writing to files, sending emails, making network calls
- Use stubs or fakes to simulate effects safely

🧱 3. Untestable Interfaces
- Hard-to-control collaborators (e.g., time, random, hardware)
- Use doubles to inject predictability

Test doubles give you speed, control, and confidence — without depending on real-world chaos.

PAUSE AND PROJECT REFLECT

Pick one recent test you wrote — How would you improve it using a test double or better assertions?

Pick one recent test a team member wrote and ask the same question. Share these answers in your next team meeting!