UNIT TESTS PASSING

NO INTEGRATION TESTS

Module 2-7

Integration Testing

Objectives

- What is an integration test?
- DAO Integration testing

Integration Testing

- Broad category of tests that validate integration between

    - Units of code

    - Outside dependencies such as databases or network resources

Integration Testing

- Use same tools as unit tests (i.e. Junit)

- Usually slower than unit tests (but still measured in ms)

- More complex to write and debug

- Can have dependencies on outside resources like files or a database

DAO Integration Testing

DAOs exist solely to interact with database
Best tested with integration tests

Rules of testing:
- DRY – production code should be DRY – don't repeat yourself
- WET – testing code should be WET – write everything twice

DAO Integration Testing

Integration tests with a database should ensure that the DAO code functions correctly:

- SELECT statements are tested by inserting dummy data before the test
- INSERT statements are tested by searching for the data
- UPDATE statements are tested by verifying dummy data has been changed
- DELETE statements are tested by seeing if dummy data is missing

DAO Integration Testing

Tests should be:
- Repeatable – If test passes/fails on first execution, it should pass/fail on second execution if no code has changed
- Independent – A test should be able to run on its own, independently of other tests, OR together with other tests and have the same result either way
- Obvious – When a test fails, it should be as obvious as possible as to why it failed

How to manage test data

- Remotely Hosted Shared Test Database
  - Advantages:
    - Easy setup
    - Production-like software and (possibly) hardware
  - Disadvantages
    - Unreliable and brittle
    - Lack of test isolation
    - Temptation to rely on existing data (which can change)

How to manage test data

- ● Locally Hosted Test Database
  - ○ Advantages
    - ■ Production-like software
    - ■ Reliable (local control)
    - ■ Isolation
  - ○ Disadvantages
    - ■ Requires local hardware resources
    - ■ RDBMS needs to be installed and managed

How to manage test data

- Embedded, In-memory Database
  - Advantages
    - Very Reliable
    - Consistent across dev machines (managed by source control)
    - Lightweight
  - Disadvantages
    - Not same software used in production
    - Cannot use proprietary features of production RDBMS

11

Mocking

- Make a replica or imitation

- Creating objects that simulate the behavior of real objects

- Typically used in unit testing, but we need to create fake data in order to test CRUD statements

Database considerations

- When testing, we create "test data"

    - Insert new data, update data, or remove rows of data
- Do not want these to be permanent changes

    - Need to roll back changes when done

SingleConnectionDataSource class

- We have used BasicDataSource for our production code
- For integration testing, we use SingleConnectionDataSource

  - Preferred implementation for testing

- Both BasicDataSource and SingleConnectionDataSource are implementations of DataSource

```
/* Using this particular implementation of DataSource so that
 * every database interaction is part of the same database
 * session and hence the same database transaction */
private SingleConnectionDataSource adminDataSource;
```

@PostConstruct method

- Generally set up the data source in a @PostConstruct method:

```
/* This method creates the temporary database to be used for the tests. */
  @PostConstruct
  public void setup() {
      if (System.getenv("DB_HOST") == null) {
          adminDataSource = new SingleConnectionDataSource();
          adminDataSource.setUrl("jdbc:postgresql://localhost:5432/postgres");
          adminDataSource.setUsername("postgres");
          adminDataSource.setPassword("postgres1");
          adminJdbcTemplate = new JdbcTemplate(adminDataSource);
          adminJdbcTemplate.update("DROP DATABASE IF EXISTS \"" + DB_NAME + "\";");
          adminJdbcTemplate.update("CREATE DATABASE \"" + DB_NAME + "\";");
      }
  }
```

https://www.baeldung.com/spring-postconstruct-predestroy

@Before method

- Where we would insert mocked data into the database:

```
@Before
public void setup() {
    sut = new JdbcCityDao(dataSource);
    testCity = new City(0, "Test City", "CC", 99, 999);
}
```

@After method

- Want to rollback after each test method runs using the @After annotation:

```java
/* After each test, we rollback any changes that were made to the database so that
 * everything is clean for the next test */
@After
public void rollback() throws SQLException {
    dataSource.getConnection().rollback();
}
```

@PreDestroy method

- Destroy the data source when done with all the tests using the @PreDestroy annotation

```java
/* This method runs after all the tests and removes the temporary database. */
@PreDestroy
public void cleanup() {
    if (adminDataSource != null) {
        adminJdbcTemplate.update("DROP DATABASE \"" + DB_NAME + "\";");
        adminDataSource.destroy();
    }
}
```

Objectives

● What is an integration test?
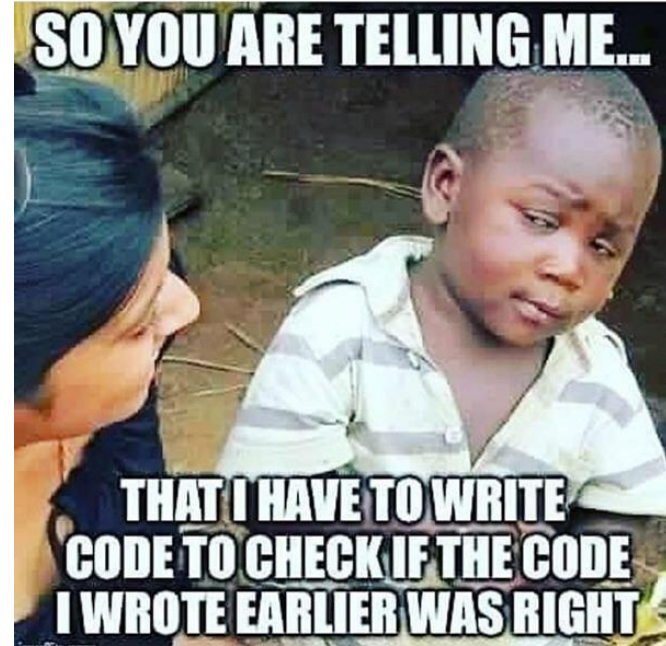
Objectives

- What is an integration test?
- DAO Integration testing

	Unit Testing	Integration Testing
(broken chain links)	✅	❌
(connected chain links)	✅	✅

Had to explain to a colleague why integration tests are important. I came up with this analogy.

Let's Code!