TESTING IN CLEAN ARCHITECTURE

CSC 207 SOFTWARE DESIGN

Computer Science
UNIVERSITY OF TORONTO

Course Overview

Tools (Weeks 1-4)

- Java
- Version Control
- Software Tools

Design (Weeks 5-8)

- **Clean Architecture**
- SOLID
- Design Patterns

Professional Topics (Weeks 9-12)

- Ethics
- Internships
- GenAI

- Last week we introduced Clean Architecture.
- This week, we will continue to explore Clean Architecture and talk about packages in Java.

Questions to be answered this week...

- How can we test a program which adheres to Clean Architecture?
- What is mocking?
- What are packages in Java?
- How can we effectively use packaging in our Java projects?

LEARNING OUTCOMES

Be able to describe several different kinds of testing

Be able to write unit tests in Java using Junit, focusing on Entities and Use Cases

Be able to write an end-to-end test, provided you have a similar example to work from

Understand the difference between unit tests, integration tests, and end-to-end tests

BEYOND SIMPLE UNIT TESTING

We talked about unit testing back in Week 4.

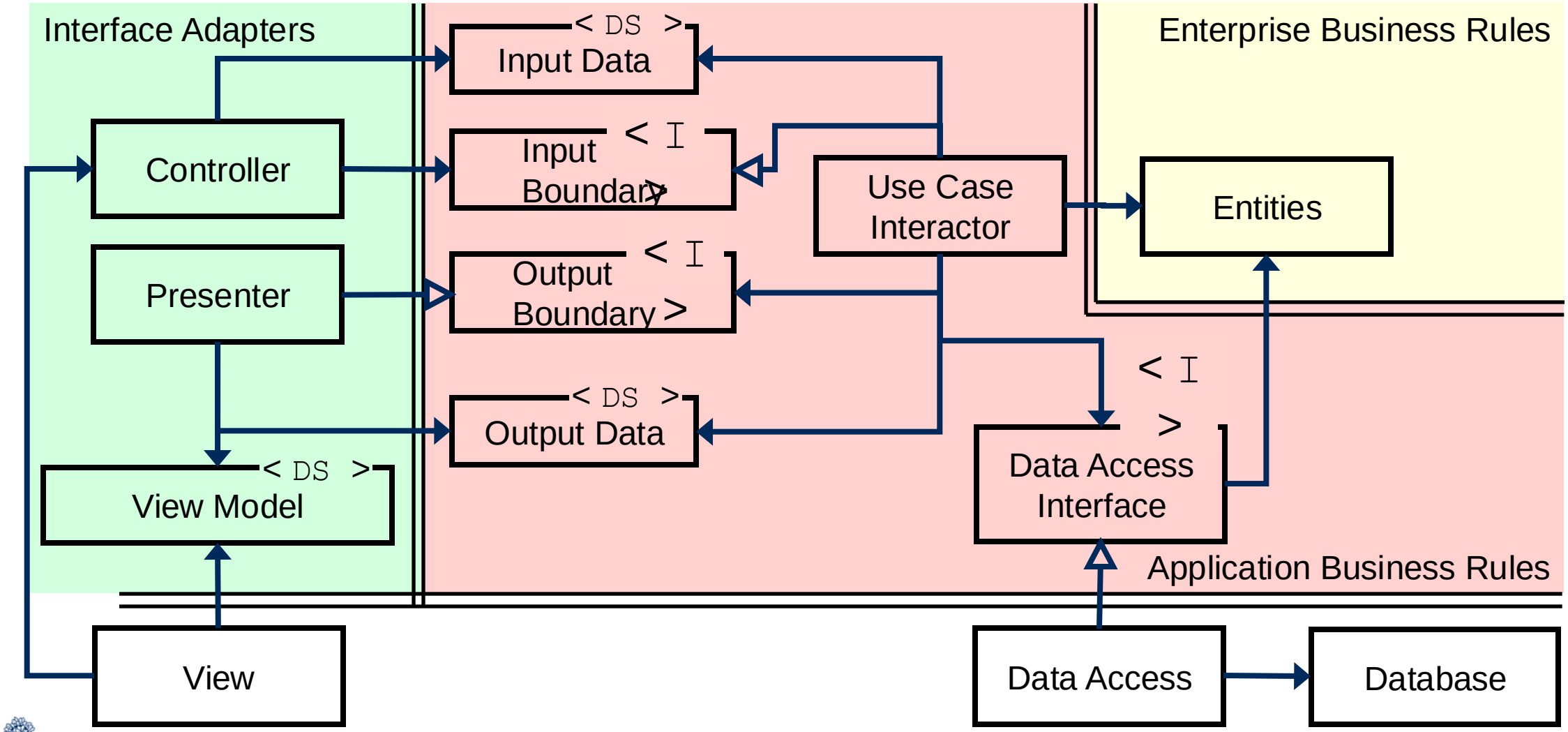
Now that we have our more complicated Clean Architecure structure to work with, how can we test our code?

For example, how can we test a UseCaseInteractor since it depends on implementations of various interfaces to function correctly?

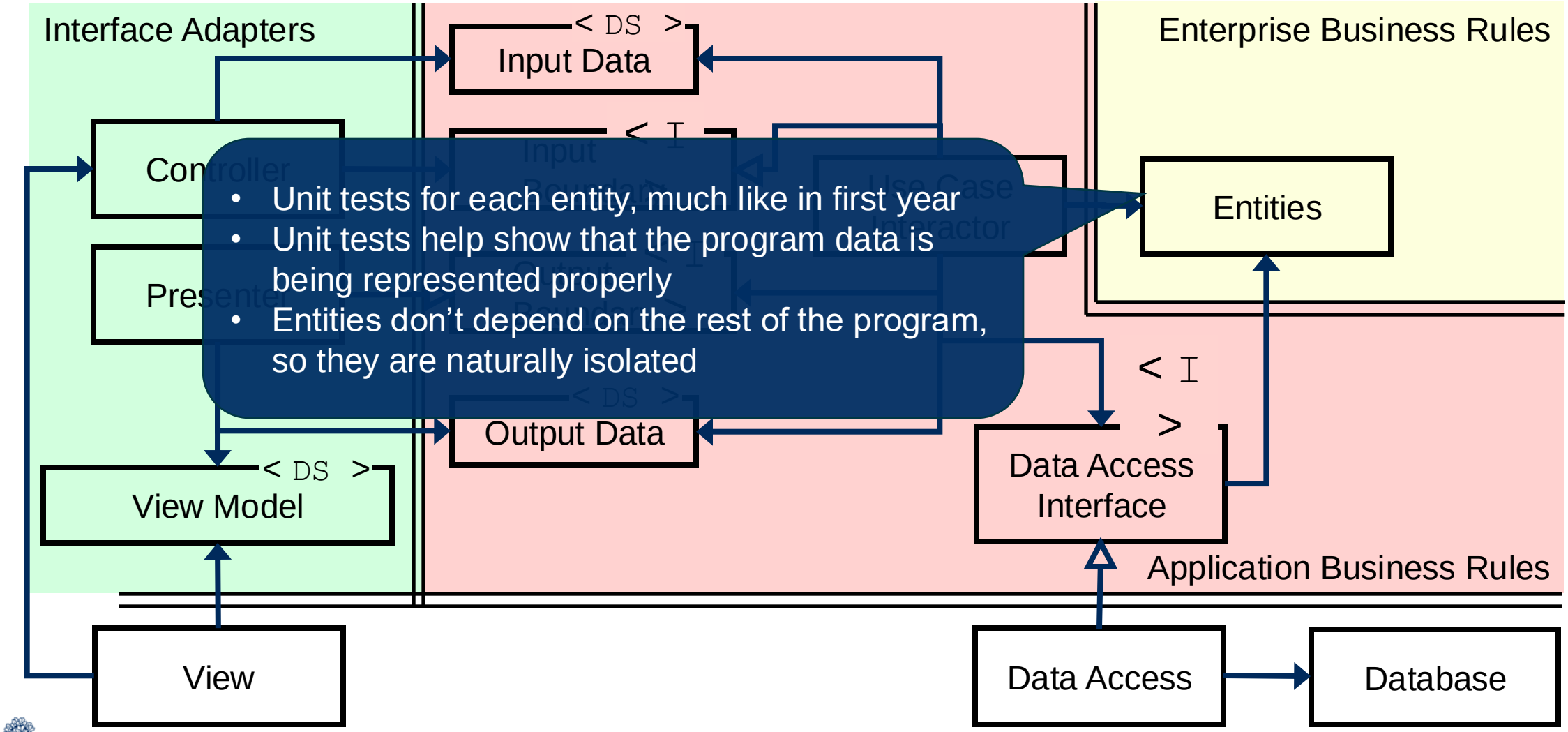
THREE TYPES OF TESTS

- Unit Test – Tests the smallest unit of cohesive code, often a method. We can create a test class called EntityTests of unit tests for all the methods in class Entity, for example.
- Integration Test – Tests how two components (say classes) interact with each other.
- End-to-end Test -- Tests an entire path through the code from input to output. This can start and end with the View.

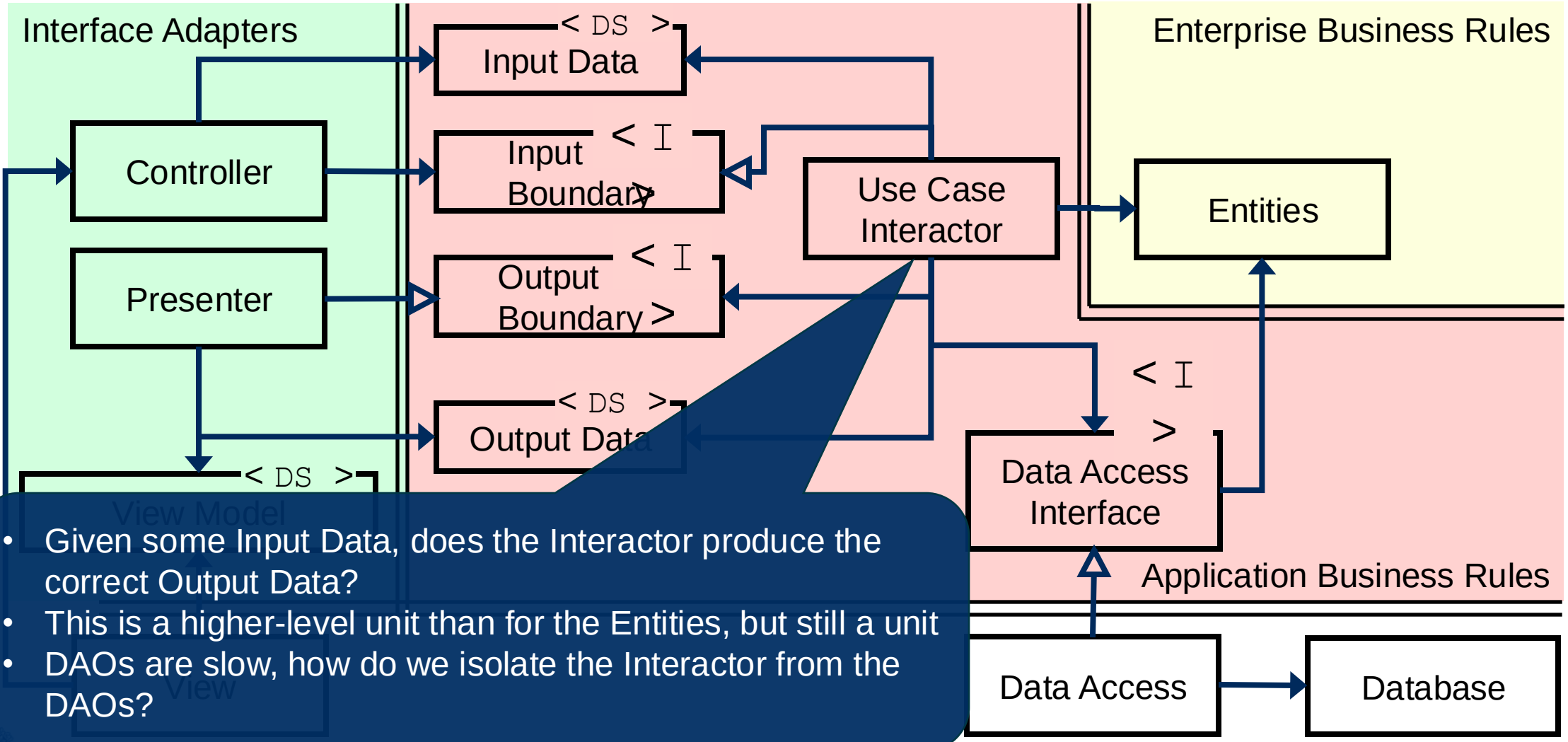
HOW CAN WE TEST THIS?



ENTITIES: UNIT TESTING

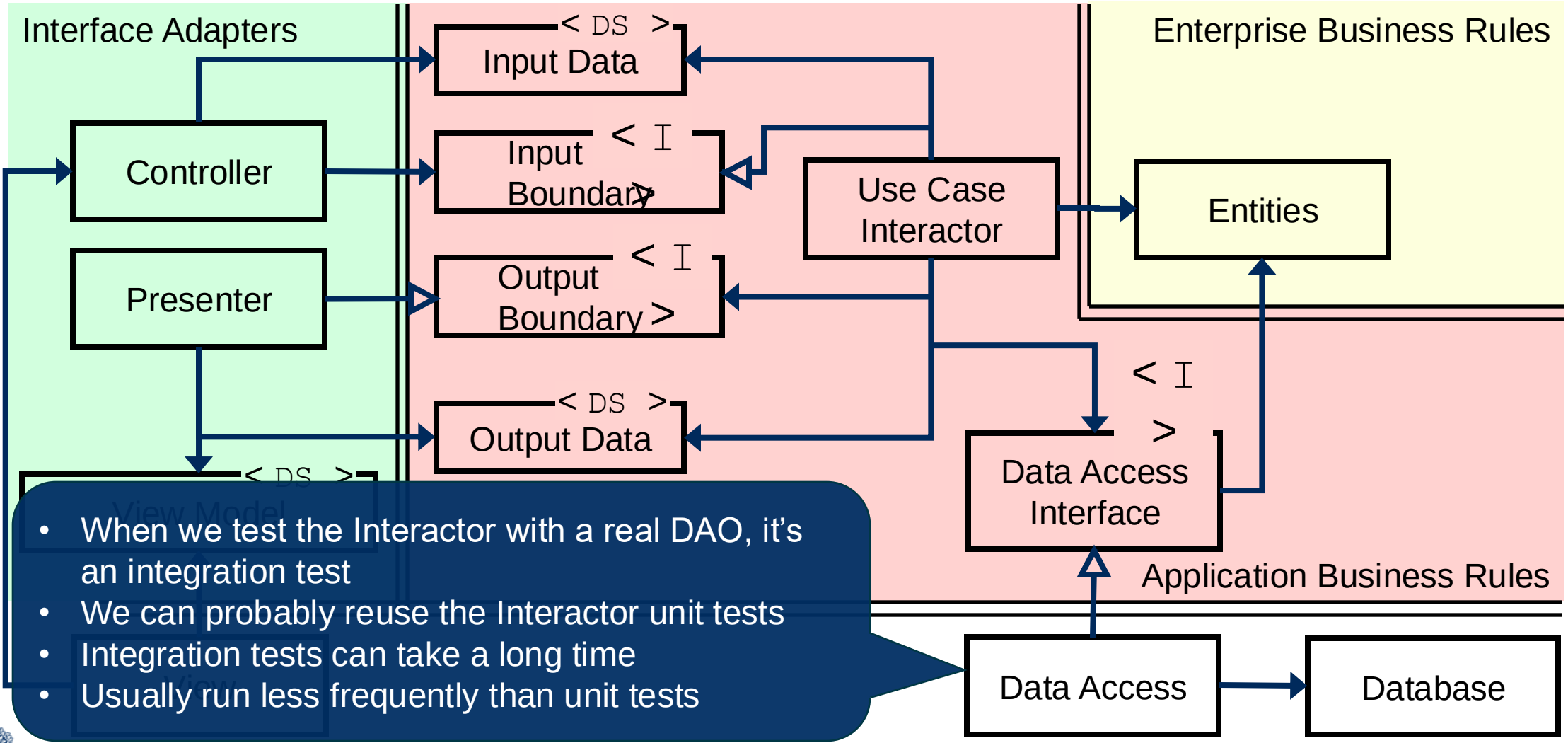


INTERACTORS: UNIT TESTING

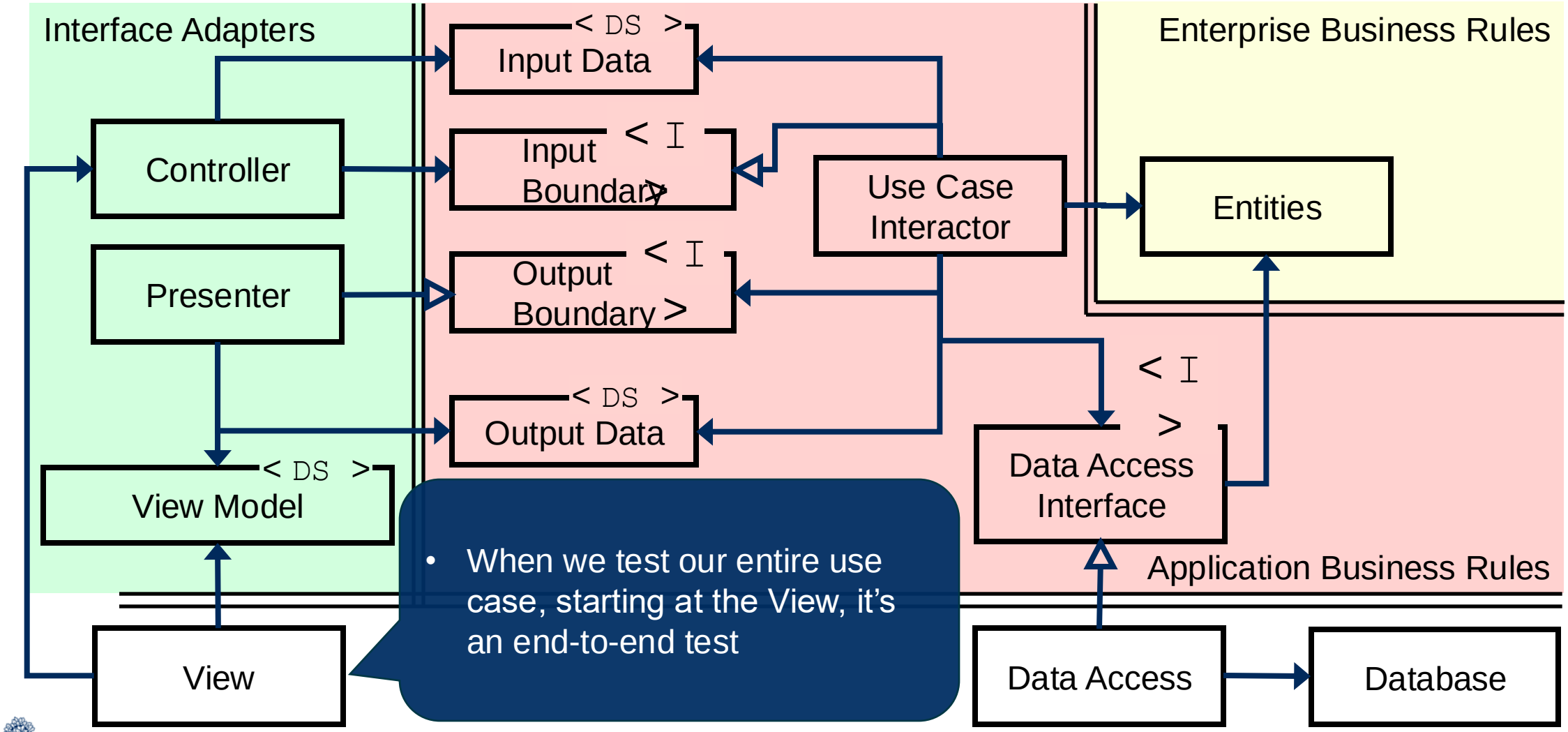


- Given some Input Data, does the Interactor produce the correct Output Data?
- This is a higher-level unit than for the Entities, but still a unit
- DAOs are slow, how do we isolate the Interactor from the DAOs?

INTERACTORS AND DAOS: INTEGRATION TESTING



HOW CAN YOU ISOLATE EACH LAYER?



DESIGN FOR TESTABILITY

- The Clean Architecture is designed to encourage testing
- The ability to mock each layer is powerful
- Refactor code if it is hard to test

USE CASE INTERACTOR TESTING

Interactors receive Input Data from a Controller and produce Output Data for a Presenter

Your test class serves as the Controller

Your test class needs to create a Presenter that checks the Output Data and any Entity and persistence mutation

In which layer should your test class go?

USE CASE INTERACTOR: THE SUCCESS TEST

```
void successTest() {
    SignupUserDataAccessInterface userRepository = new InMemoryUserDataAccessObject();

    SignupOutputBoundary successPresenter = // Make a presenter here that asserts things

    SignupInputData inputData = new SignupInputData("Paul", "password", "password");
    SignupInputBoundary interactor = new SignupInteractor(
            userRepository, successPresenter, new CommonUserFactory());
    interactor.execute(inputData); // This will eventually send Output Data to the successPresenter
}
```

USE CASE INTERACTOR: THE PRESENTER

This goes in the successTest method:

```
// This instantiates an anonymous SignupOutputBoundary implementing class
SignupOutputBoundary successPresenter = new SignupOutputBoundary() {
    @Override
    public void prepareSuccessView(SignupOutputData user) {
        // 2 things to check: the output data is correct, and the user has been created in the DAO.
        assertEquals("Paul", user.getUsername());
        assertTrue(userRepository.existsByName("Paul"));
    }

    @Override
    public void prepareFailView(String error) {
        fail("Use case failure is unexpected.");
    }
};
```

EXTRA ADVICE

- If you want to test the same use case with different inputs, you may want to construct the necessary parts of the CA engine once in a @BeforeAll method or make use of @BeforeEach.
- Don't forget to test for both success and failures!
- Taking the time to write these tests for each use case as you develop the use case will help ensure the quality and correctness of your final code.
- In a team environment, expect your team members to write tests for any code they are submitting a pull request for before you will approve their request!

LET'S GO LOOK AT TESTS IN INTELLIJ!

The Homework 5 starter code:

- https://github.com/CSC207-2024F-UofT/CAUserLogin.git