Developing a Unit Test for a Use Case Interactor

This series of activities will explore how to create a unit test for a use case interactor. Here is a new user story for the ca-user-login project:

As an admin, I want to find out if a user account is new (less than 24 hours old) so that I can send them a personal welcome message.

On the back of this handout is a UML class diagram for a use case interactor that supports this user story. Note that we now assume that the creation date of a user is saved as part of our `User` entity.

Part I

Understanding the UML diagram

Activity 1 Here are some statements about the UML diagram.

i. The most important class is `NewAccountInteractor`. **All the other `New*` interfaces and classes exist purely to support the interactor.**

ii. The Input Boundary exists so that the controller knows how to invoke the interactor.

iii. The Input Data class exists to package up the data the interactor needs to do its work. The Controller instantiates it.

iv. The `NewAccountDataAccessInterface` is implemented by a Data Access Object (DAO). This interface exists so that the DAO knows what methods the interactor will use to request Entities from the data layer.

v. The Output Boundary exists so that the interactor knows what methods it can call to deliver the output data to the UI layer. A Presenter implements this interface.

vi. The Output Data class exists to package up the data the interactor produces to send to the Presenter. The interactor instantiates the Output Data.

Which statements are correct?

A None of them.

B Only i, ii, and v.

C Only i, ii, iii, v, and vi.

D Only ii, iii, iv, v, and vi.

E All of them.

Part II
Input and output of the interactor

For each activity, select all that apply.

Activity 1 What information does the use case interactor expect to receive?

A The account creation time.

B The input boundary.

C The username of the account to check.

D A brand new account.

Activity 2 What information does the use case interactor produce during the main flow?

A Whether the account is new (less than 24 hours old).

B The username of the account.

C The `NewAccountPresenter` object.

D An object containing all the information that needs to be updated in the UI.

Activity 3 To make this use case run, you need to create a `NewAccountInputData` object and pass it to the input boundary's `execute` method. Which code fragments do those things?

A
```
NewAccountInputData userAccountInputData = new NewAccountInputData("paul");
interactor.execute(userAccountInputData);
```

B
```
interactor.execute(new NewAccountInputData("paul"));
```

C
```
String username = "paul";
userAccountInputData.execute(interactor, username);
```

D
```
String userAccountInputData = new String("paul");
interactor.execute(userAccountInputData);
```

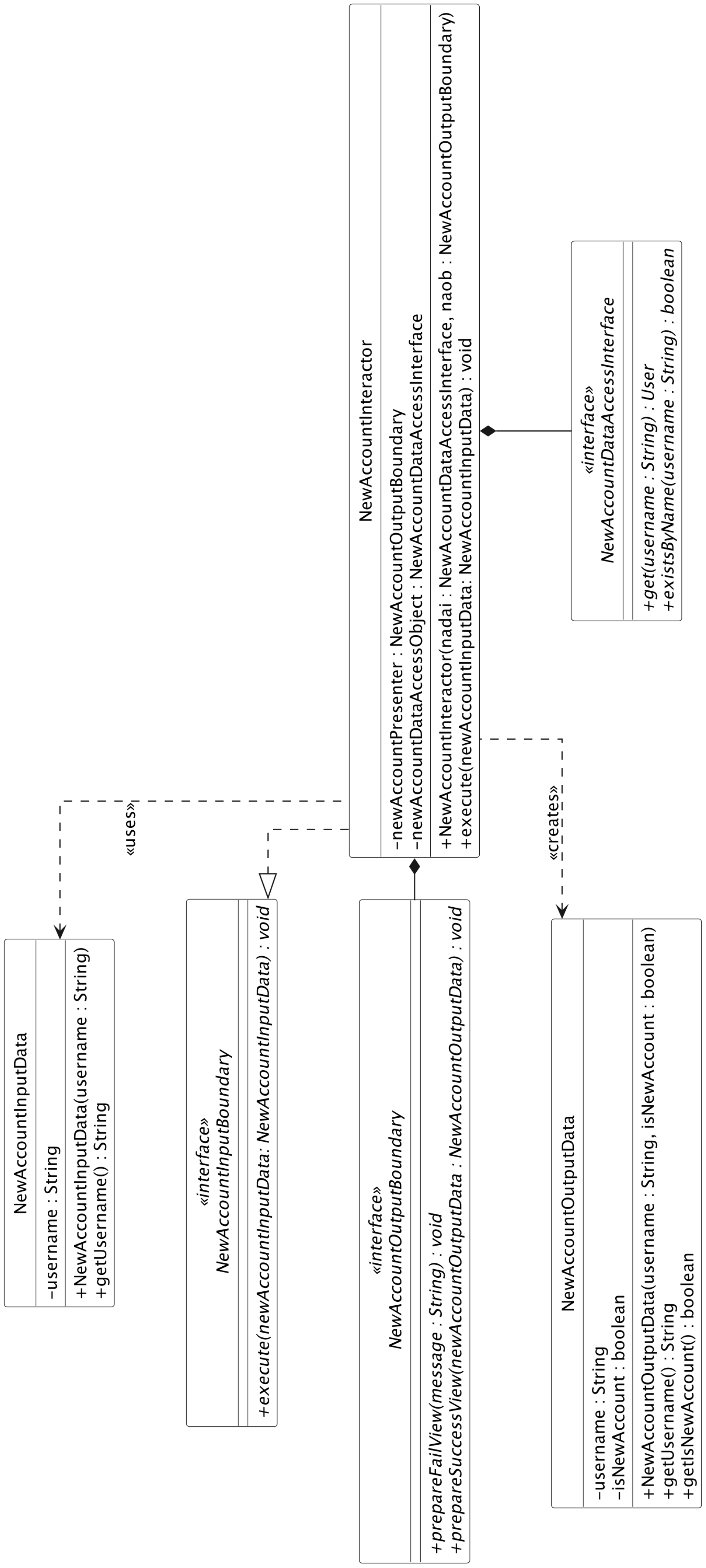
Activity 4 When the interactor's work is complete, it creates an instance of `NewAccountOutputData`. What values will be assigned to instance variables `username` and `isNewAccount`, assuming you're using the code fragment from the previous Activity?

A Exactly the same information as `NewAccountInputData`.

B The code will crash because the user does not exist.

C `"paul"` and `true`

D `"paul"` and `false`


NewAccountInteractor
−newAccountPresenter : NewAccountOutputBoundary
−newAccountDataAccessObject : NewAccountDataAccessInterface
+NewAccountInteractor(nadai : NewAccountDataAccessInterface, naob : NewAccountOutputBoundary)
+execute(newAccountInputData: NewAccountInputData) : void
«interface»
NewAccountDataAccessInterface
+get(username : String) : User
+existsByName(username : String) : boolean
«uses»
«creates»
NewAccountInputData
−username : String
+NewAccountInputData(username : String)
+getUsername() : String
«interface»
NewAccountInputBoundary
+execute(newAccountInputData: NewAccountInputData) : void
«interface»
NewAccountOutputBoundary
+prepareFailView(message : String) : void
+prepareSuccessView(newAccountOutputData : NewAccountOutputData) : void
NewAccountOutputData
−username : String
−isNewAccount : boolean
+NewAccountOutputData(username : String, isNewAccount : boolean)
+getUsername() : String
+getIsNewAccount() : boolean

Part III
What needs to happen to test this?

Activity 1 Remember the goal today is to write a unit test for the interactor, and that we are testing that a new account is handled properly. To do this, you will need to provide the interactor with:

- A Data Access Object that implements the ______________________________ interface.
- A _____________ that implements the `NewAccountOutputBoundary` interface.
- An instance of ____________________ containing the username of the account to check.

A `NewAccountInputBoundary`, View, `NewAccountOutputData`, `prepareFailView`

B `NewAccountDataAccessInterface`, Presenter, `NewAccountInputData`

C `NewAccountDataAccessInterface`, Controller, `NewAccountOutputData`

D `NewAccountInputBoundary`, Presenter, `NewAccountInputData`

Activity 2 This code creates an interactor and executes it:

```
NewAccountInputBoundary interactor = new NewAccountInteractor(userRepository,
                                                              successPresenter);
interactor.execute(inputData);
```

- Veriable `userRepository` refers to a `NewAccountDataAccessInterface`. This is the DAO. It's a *mock* DAO because it is an in-memory implementation used only for testing. It uses a hashmap to store users. It doesn't write to a file or a database.
- Variable `successPresenter` refers to a `NewAccountOutputBoundary`.
- Variable `inputData` refers to an instance of `NewAccountInputData`.

Which lines of code are necessary to initialize `userRepository`, `successPresenter`, and `inputData` for the unit test? The next page contains the code for the `createTestUser` and `createTestPresenter` methods.

A
```
NewAccountInputData inputData = new NewAccountInputData("paul");
NewAccountDataAccessInterface userRepository = new NewAccountDataAccessInterface();
createTestUser(userRepository);
NewAccountOutputBoundary successPresenter = new NewAccountOutputBoundary();
```

B
```
NewAccountInputData inputData = new NewAccountInputData("paul");
InMemoryUserDataAccessObject userRepository = new InMemoryUserDataAccessObject();
NewAccountOutputBoundary successPresenter = createTestPresenter();
```

C
```
NewAccountInputData inputData = new NewAccountInputData("paul");
InMemoryUserDataAccessObject userRepository = new InMemoryUserDataAccessObject();
createTestUser(userRepository);
NewAccountOutputBoundary successPresenter = createTestPresenter();
```

D
```
NewAccountInputData inputData = new NewAccountInputData("paul");
NewAccountDataAccessInterface userRepository = new NewAccountDataAccessInterface();
NewAccountOutputBoundary successPresenter = new NewAccountOutputBoundary();
```

The following method creates the Presenter for the unit test. Notice that it is an anonymous class, much like the `ActionListeners` in a UI. Notice the Assertions in the Presenter's `prepareSuccessView` method to check that the Output Data is correct.

```
    private static NewAccountOutputBoundary createTestPresenter() {
        return new NewAccountOutputBoundary() {
            @Override
            public void prepareSuccessView(NewAccountOutputData user) {
                assertEquals("paul", user.getUsername());
                assertTrue(user.getIsNewAccount());
            }

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

This method creates a new test user and saves it to the provided user repository.

```
    private static void createTestUser(InMemoryUserDataAccessObject userRepository) {
       UserFactory userFactory = new UserFactory();
        LocalDateTime now = LocalDateTime.now();
        User user = userFactory.create("paul", "Iforgot", now);
        userRepository.save(user);
    }
```