JUnit

EXERCISE BOOK

**Software Developer Level 4: Module Three**

**Contents**

[Exercise 1 – Testing existing code 3](#_Toc124253118)

[a) Create a test plan 3](#_Toc124253119)

[b) Implement your test plan 3](#_Toc124253120)

[Exercise 2 – Testing exceptions 4](#_Toc124253121)

[a) Create a test plan 4](#_Toc124253122)

[b) Implement your test plan 5](#_Toc124253123)

[Exercise 3 – Mocking in a unit test 6](#_Toc124253124)

[a) Update the test plan from exercise 2 6](#_Toc124253125)

[b) Implement the tests 6](#_Toc124253126)

[c) Test driven development (Stretch task) 7](#_Toc124253127)

# Exercise 1 – Testing existing code

This exercise uses the **Calculator** class found in the **exercise1** package of this repository. Clone the repository and import the project into Eclipse to get started.

**Exercise**: [MrWalshyType2/QAA-Module3-UnitTest-Exercises (github.com)](https://github.com/MrWalshyType2/QAA-Module3-UnitTest-Exercises)

**Solution**: [QAA-Module3-UnitTest-Exercise-Solutions/CalculatorTest.java at main · MrWalshyType2/QAA-Module3-UnitTest-Exercise-Solutions (github.com)](https://github.com/MrWalshyType2/QAA-Module3-UnitTest-Exercise-Solutions/blob/main/src/test/java/com/qaa/module3/unit_testing_exercises/exercise1/CalculatorTest.java)

## Create a test plan

In this exercise, you are required to create a test plan which consists of test cases for the **Calculator** classes 4 methods. Use the following template for creating your test cases:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Method | Description | Inputs | Expected output | Actual output |
| 1 | add(double num1, double num2) | Adding two small numbers | num1=10  num2=30 | 40 |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

One test case has been created for you as an example. It is expected that you produce at least 3 test cases per method:

* Test borderline input values, i.e., what are the highest values you can add, what about the smallest…
* Test at least 1 normal input combination

## Implement your test plan

The **Calculator** class has already been created, use your test plan to guide the development of tests for the methods of this class.

# Exercise 2 – Testing exceptions

This exercise uses the **UserService** class defined in the **exercise2** package of the repository. Clone the repository and import the project into Eclipse to get started.

**Repository**: [MrWalshyType2/QAA-Module3-UnitTest-Exercises (github.com)](https://github.com/MrWalshyType2/QAA-Module3-UnitTest-Exercises)

**Solution:** [QAA-Module3-UnitTest-Exercise-Solutions/UserServiceTest.java at main · MrWalshyType2/QAA-Module3-UnitTest-Exercise-Solutions (github.com)](https://github.com/MrWalshyType2/QAA-Module3-UnitTest-Exercise-Solutions/blob/main/src/test/java/com/qaa/module3/unit_testing_exercises/exercise2/UserServiceTest.java)

## Create a test plan

In this exercise, you are required to create a test plan which consists of test cases for the **UserService** classes 2 methods. Use the following template for creating your test cases:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Method** | **Description** | **Inputs** | **Expected output** | **Actual output** |
| 1 | login(String username, String password) | Register a valid user, login successfully with said valid user. | login   * username=”bobby” * password=”Codes123”   register   * username=”bobby” * password=”Codes123” | “bobby” |  |
| 2 | register(String username, String password) | Register a user with an invalid password due to missing number. | register   * username=”bobby” * password=”Codes” | IllegalArgumentException (“Password must contain at least 1 number character”) |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Two example test cases have been created for you. It is expected that you produce a test case for every possible exception that could be thrown.

## Implement your test plan

The **UserService** class has already been created, use your test plan to guide the development of tests for the methods of this class.

# Exercise 3 – Mocking in a unit test

This exercise relies on the **User**, **UserController** classes and **UserRepository** interface in the **exercise3** package in the supplied repository.

**Repository**: [MrWalshyType2/QAA-Module3-UnitTest-Exercises (github.com)](https://github.com/MrWalshyType2/QAA-Module3-UnitTest-Exercises)

## Update the test plan from exercise 2

The test plan from exercise 2 can be reused for this example. Modify the test plan to accommodate the changes to the **login** and **register** methods present in the **UserController** class.

* As we are now dealing with multiple classes, it is also recommended to add a **Class** column to the test table
* There is a new exception that could be thrown in the **register** method
* Some exceptions have been removed from the **login** method as it is expected that the repository implementation would handle those cases in this example, i.e., invalid usernames or passwords

## Implement the tests

Implement your unit test plan, as done with the previous examples.

* Be careful when writing your tests for the **login** and **register** methods, it is expected that you use the Mockito framework to mock interactions with the repository
* Make sure to mock the repository and inject it into the controller with the @Mock and @InjectMocks annotations respectively
* The **repository** methods being mocked are: **UserRepository.exists()**, **UserRepository.register()** and **UserRepository.login()**.

## Test driven development (Stretch task)

If you complete the above task, create a test plan for the methods of the **UserRepository** interface.

Once a suitable plan is created, create your tests and implement the interface as a class, call it **ConcreteUserRepository**.

* The **Concrete** in the name indicates that it is a class and not an interface or abstract class.
* Store the instances of **User** in a **List<User>** instance variable on the concrete repository class

Advice:

1. After creating the plan, create the concrete repository class and implement the **UserRepository** interface.
2. Add the empty method stubs
3. Create the test class **UserRepositoryTest**
4. Start creating the **register** tests
5. Create the implementation of **ConcreteUserRepository.register()** as you write the test
6. Repeat steps 4 and 5 for the **login** and **exists** methods