Unit Testing with Mocking

Soure Code Repo: **https://github.com/Narsi-Myteaching/Junit-Mockito-Workshop-January-2025.git**

Agenda:

Day-1:

1. What is Unit Testing?

2. What is Unit?

3. What is Junit?

4. Why Unit tesing?

5. Principles of Unit testing

6. System Under Test (SUT)

7. Method Under Test (MUT)

8. Testing Pyramid

9. What is Junit5?

10. Junit5 Architecture.

Day-2:

1. Unit Tetsing Env Setup

2. Checkstyle

3. TDD

4. Practically demonistrate the TDD

5. Introcution Mockito.

6. What is Mock and Stub?

7. @InjectMocks and @Mock annotations

8. Test Structure

9. Stubbing and Expectations

10. Argument Matchers

11. Stubbing Verification

12. Jacoco Code Coverage

1. What is Unit Testing?

Unit Test is a very simple, self contained, isolated java piece of code, that we will write to test another piece of java code.

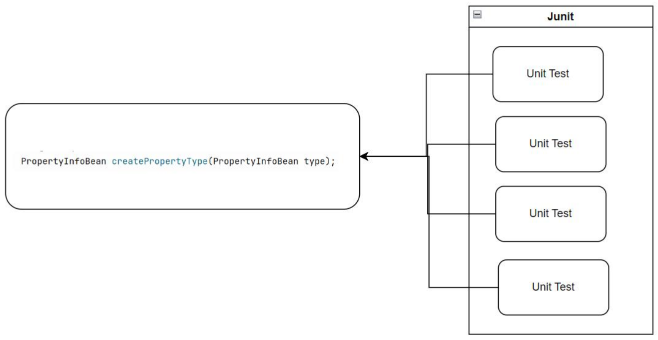
Lets say we have a java method:

```
public  booleans isValidEmail( String email) {

            // Email Validation Logic.

}
```

To ensure, this simple java code works as expected, we will write one or more unit tests.



```
PropertyInfoBean createPropertyType(PropertyInfoBean type);
```

4. Why unit testing?

Unit test is a java code that we will write to test another piece of java code.

1. Code should work:

- ✓ Java Method should work for both the valid and invalid input parameters
- ✓ Java Method should gives the same result if we execute it on other computers as well.

To ensure these, we have to write the one or more Unit Tests, where one will test the java code with the valid inputer paramaters and others may test the java code with the invalid parameters.

2. Code should work as expected with the valid and invalid parameters

Before deploying the code in other environments, developer should be enough confident that the unit method should work as expected for both valid and invalid scenarios.

In TDD, we will write the tests before the actual code implementation. So if we will write the code which will works as expected for both valid and invalid scenarios then , the design of the system is good.

3. **Code should work now and in Future**

Lets say we have implemented a complex feature which will be used by the other developers in develping their feature code.

Whats the guarantee that the existed code shoulnt break, due to the other feature implementations.

So if we write the Unit Tests, the unit teste will early detect the problem in the code and there will be chance of acting on the pronlme immediately to resolve and to ensure the existed code shoukdnt break.

4. Future code should work if the existed code updates.

If any updates happened to the existed complex code, then the feature developed code shouldn't break.

So if we write the unit test for the updated existing code, then we can

Characterstics of Unit Testing

to ensure the effective Unit Tests, we should have to follow the prinicple: **F.I.R.S.T**

F.I.R.S.T is a compbination of the following principles:

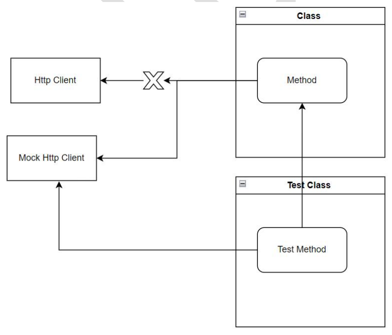
1. **F**aster: Unit Tests should execute faster.

To achive this, we have to make sure, our unit tests should not communicate with the network or should not communicate with the database.

Hence the network connection time would save as there is not network or no driver communication. It should be a plan java code to test the java method that we want with the known input and known out puts. Unit Tests shouldn't communicate with the real servers or external systems.

2. Isoltaion:

Unit test should isolate the actual java method that we want to test from its external dependencies.



3. Repeatative:

Unit tests should execute **repeataedly**,

We gonna write one ore more unit tests, for a single feature of the actual java method. That means the actual java method is executing repeatadly for the different behaviours I.e. different input parameters.

4. Self Validation:

When a unit test executes then it will produce the result of the actual java method execution either pass or fail. Once after the validation then the results will be erased for the next round of validation.

5. Thourough and Timely:

The actual java method should work as expected for botth the happy and the negative scenarios. That means we have to write one or more unit tests for achieve the desired result.

If the scenarios are more, then we also have to write more unit tests.this is called as thourough.

Timely:

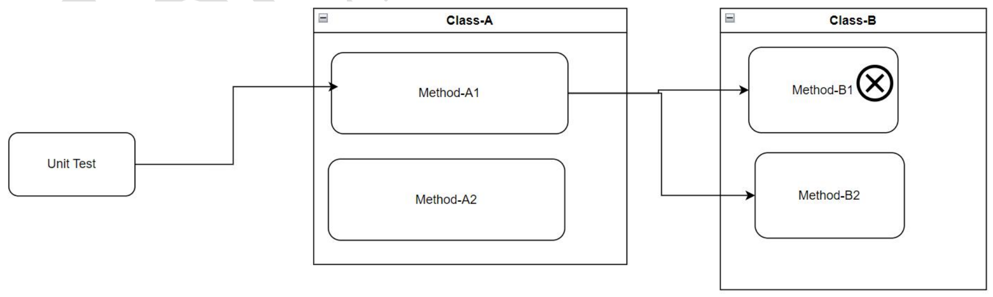
Write the unit tests at the time of feature implementation, that is even before the feature implementatiuon, write the usints. This will gives the good design of the system.

11. System Under Test (SUT)

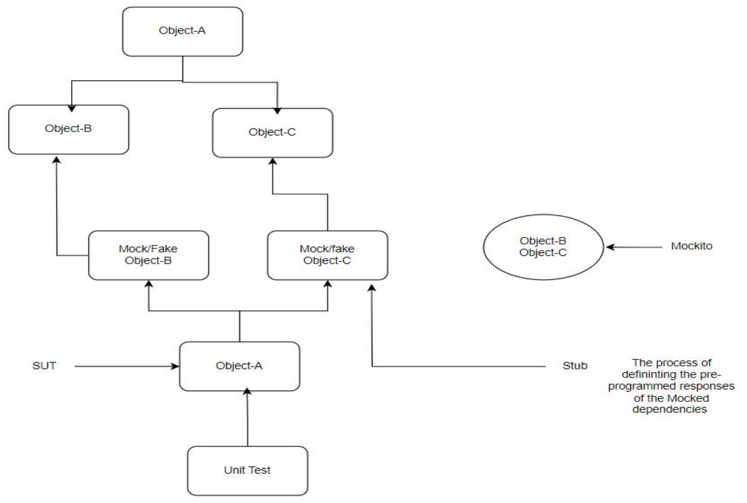
System undet Test is a specific piece of code that can **isloate and test**.

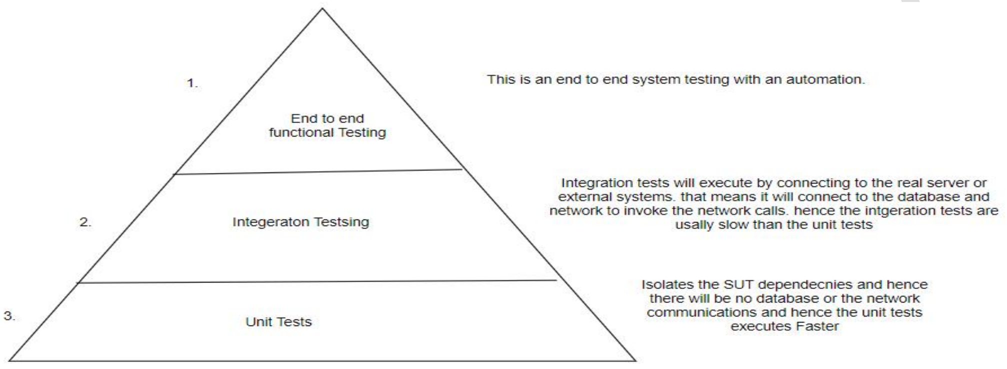
SUT can be:

- A class
- A group of classes
- A method in a class

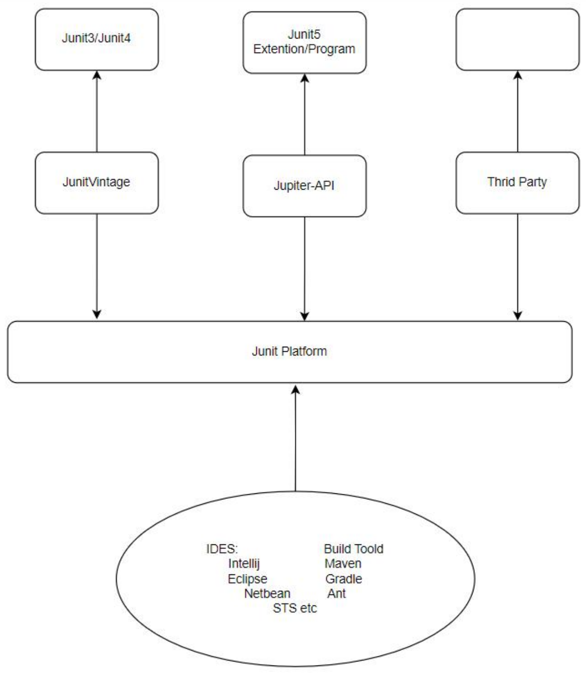


Unit Testing with Mocking




1.
End to end functional Testing
This is an end to end system testing with an automation.
2.
Integeraton Testsing
Integration tests will execute by connecting to the real server or external systems. that means it will connect to the database and network to invoke the network calls. hence the intgeration tests are usally slow than the unit tests
3.
Unit Tests
Isolates the SUT dependecnies and hence there will be no database or the network communications and hence the unit tests executes Faster

Junit5 Architecture



Unit Testing with Mocking

What is Mockito?

Mockito is a testing framework crated for testing the java methods, it allows us to test the java method by creating the “Test Doubles”.

What is Test Double?

Test Doble is a mock object that stand in before Real Objects.

- ✓ Mock
- ✓ Stub
- ✓ Spy
- ✓ Fake

Are said to be the Test Doubles.

What is Test Structure?

1. Name: test_mut_condition_result

2. AAA-Arrange-Act-Assert: is pattern that ensures the code speartion from the before execution , execution of test and the after execution of test case.

Wehat ios the input

What the SUT/MUT?

What is the result?

What is Stub?

Stub is a process of pre-programming the responses of the mocked dependencies.

Sysntax of the stub:

When(condition).thenReturns(expectation)

When: represents the condition under which stub can activate.