JUnit 5

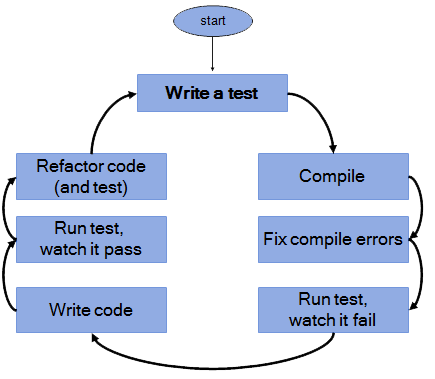
[**https://junit.org/junit5/docs/current/user-guide/**](https://junit.org/junit5/docs/current/user-guide/)

* + 1. **BASIC CONCEPT**

**JUnit** is an open source framework use for unit testing in Java programming language.

**JUnit** used to test the application without connect to database, which is take must time to setup.

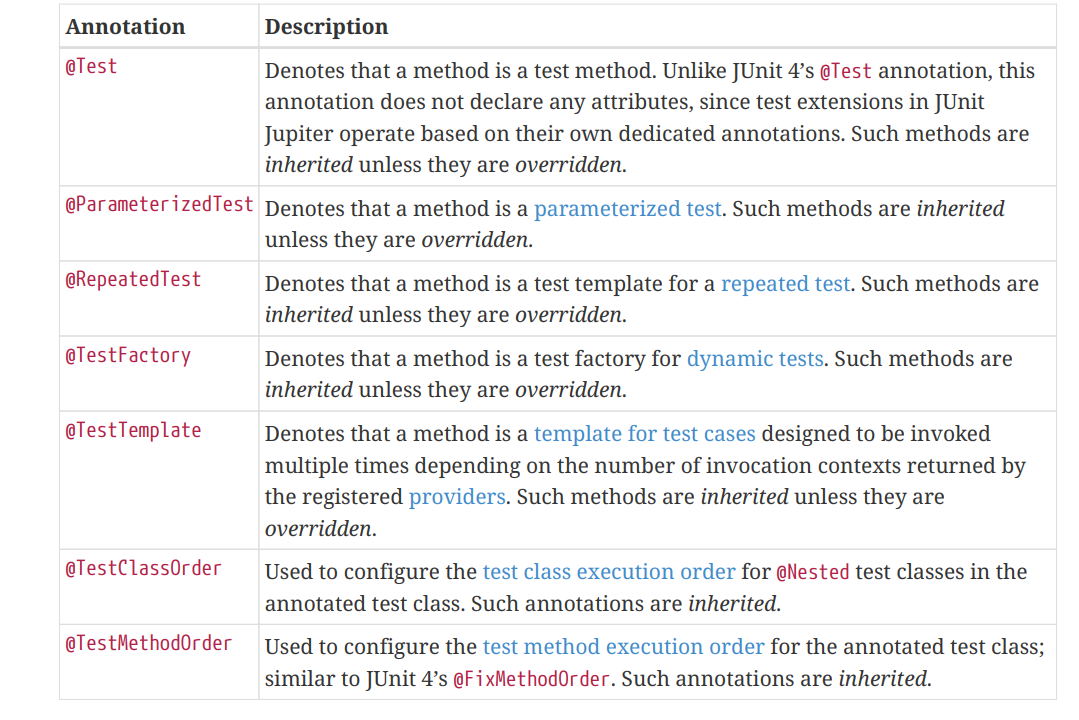
**JUnit** promotes the idea of “First testing then coding”, which means setting up the test data for a piece of code that can be tested first then implemented.

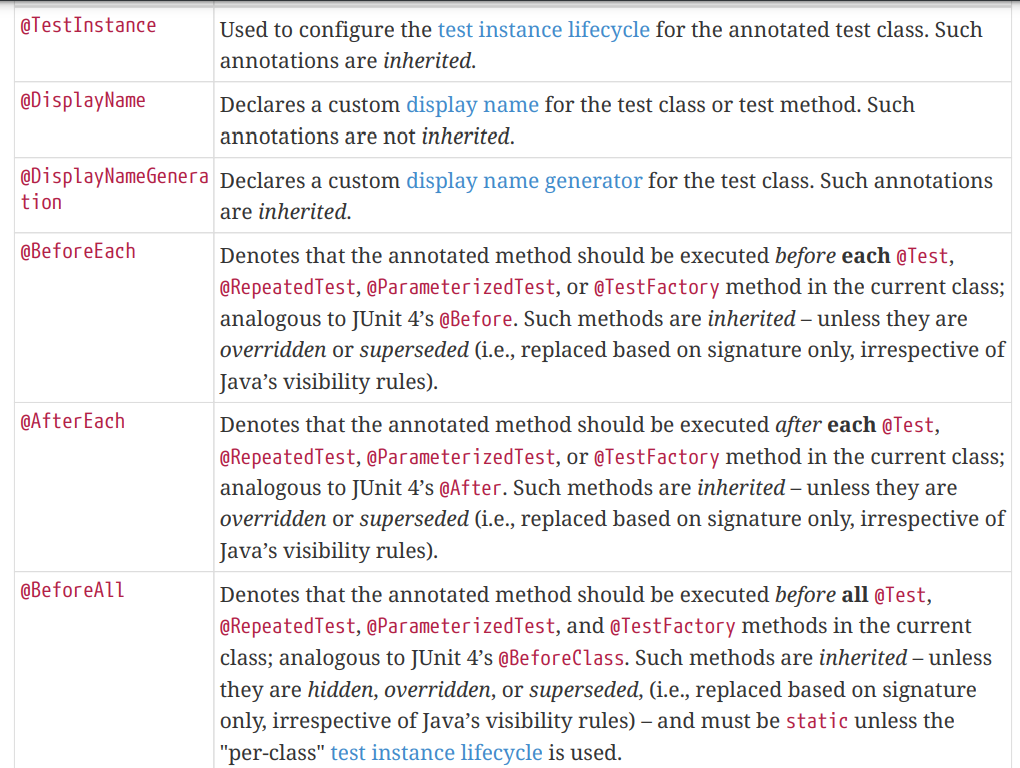


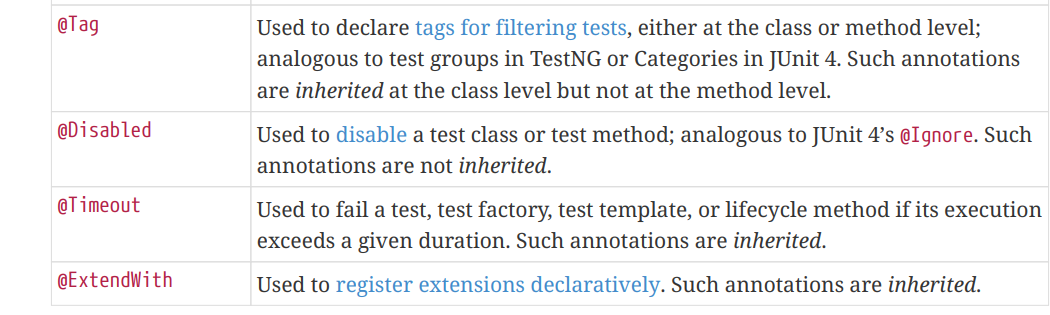
This approach is like “test a little, code a little, test a little then code a little”.

The main approach of **JUnit** is to compare the expected output with the actual result. In **JUnit**, this process is called as Assertion.

**JUnit 5 = JUnit Platform + JUnit Jupiter + JUnit Vintage**JUnit 5 requires Java 8 (or higher) at runtime. However, you can still test code that has been compiled with previous versions of the JDK.

1. **Annotations**





**Standard Test:**



**Running Tests**

In order to use a different JUnit 5 version (e.g., 5.9.3), you may need to include the corresponding versions of the junit-platform-launcher, junit-jupiter-engine, and junit-vintage-engine JARs in the classpath.





**Mockito Framework**<https://site.mockito.org/>

**Mocking**

**What is Mocking ?**

- Mocking is a testing technique where mock objects are used instead of real objects for testing purposes. Mock objects provide a specific (dummy) output for a particular (dummy) input passed to it.

- The mocking technique is not only used in Java but also used in any object-oriented programming language. There are many frameworks available in Java for mocking, but **Mockito** is the most popular framework among them.

**Why use Mocking ?**

**-** If we want to test a component that depends on the other component

**-** If the real components perform slow operations while dealing with database connections or another complex read/ write operation

**Mockito**

**What is Mockito ?**

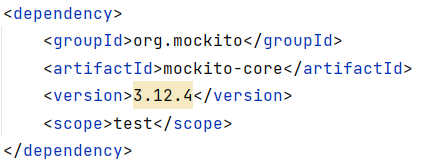
Mockito is a popular mock framework which can be used in conjunction with JUnit. Mockito allows you to create and configure mock objects. Using Mockito simplifies the development of tests for classes with external dependencies significantly.

(Using version 3.12.4 on 08/26/2021)

**Set up**

**-** Step 1: Create maven project

- Step 2: Add dependencies into pom.xml (make sure you have junit test 5 dependency before)

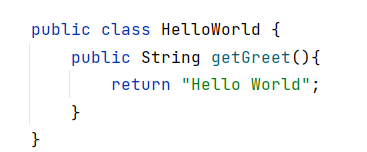




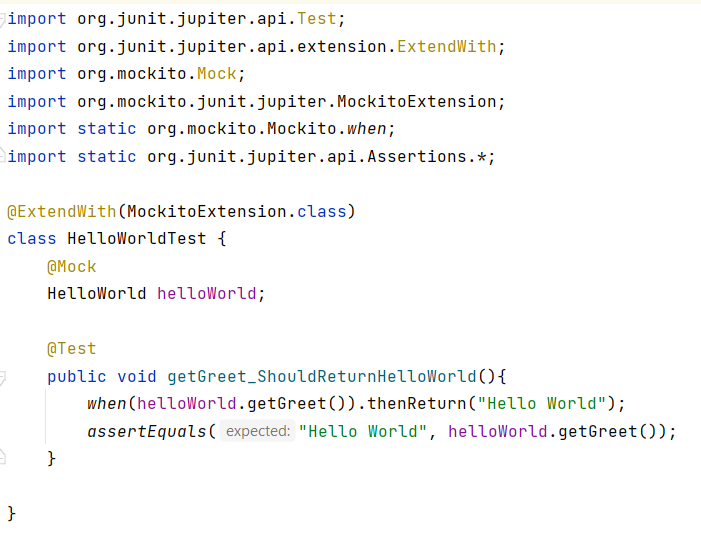
**Let’s practice with Mockito**

**1. @Mock, when(), thenReturn()**

I have a HelloWorld class like this, very simple



I create test class for HelloWorld:



Now, let’s analyze the code:

* @Mock: Use to create a mock object of HelloWorld.
* *when*(helloWorld.getGreet()).thenReturn("Hello World"): we use it when we want the mock object of HelloWorld return “Hello World” when method getGreet() is called.
* *assertEquals*("Hello World", helloWorld.getGreet()): we run test for method getGreet() with expected value is “Hello World”.
* @ExtendWith(MockitoExtension.class): this is an annotation of mockito-junit-jupiter dependency we have added before. Allow us integrate Junit 5 and Mockito. You can remove mockito-junit-jupiter dependency from pom.xml, but you must declare mock object by another way:

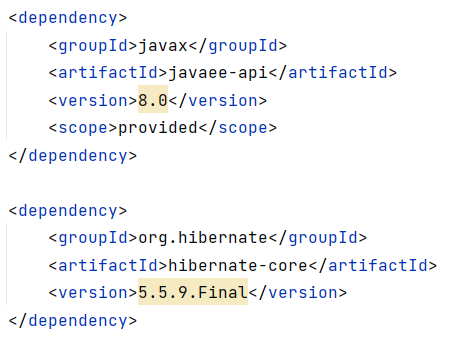


**2. @Mock, @InjectMocks, when(), thenReturn(), verify(), times()**

Now, we will try with a little more complicated case: I have pushed a sample project in github

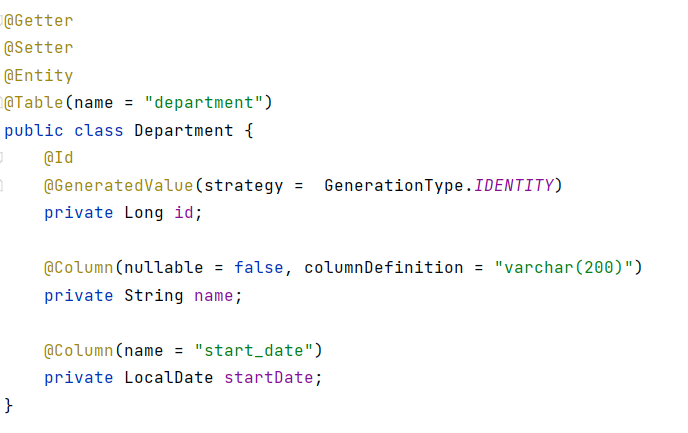
You can pull it and try. (https://github.com/thanhlong320/Mockito)

Or another way, you can add dependency javaee-api and hibernate-core.

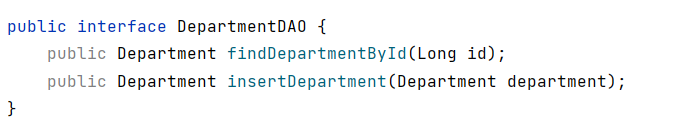


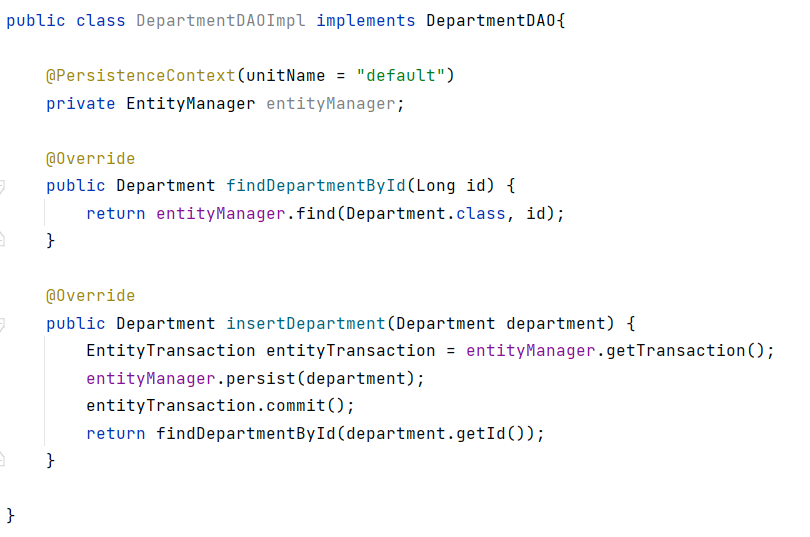
Actually, Mockito will run with mock object, so we can no need a real entity, or connection with database, but in this case, I try to make the application as real as possible.

Assume I have a Department entity:

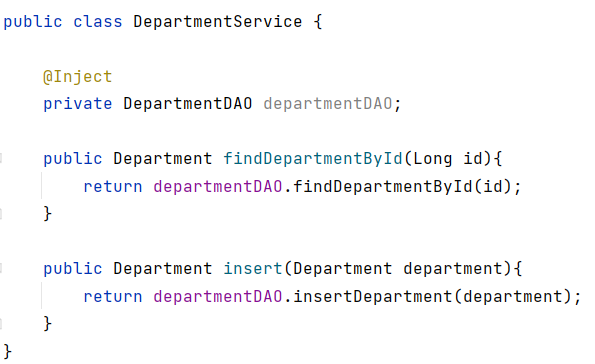


and an interface and a class implement this interface:

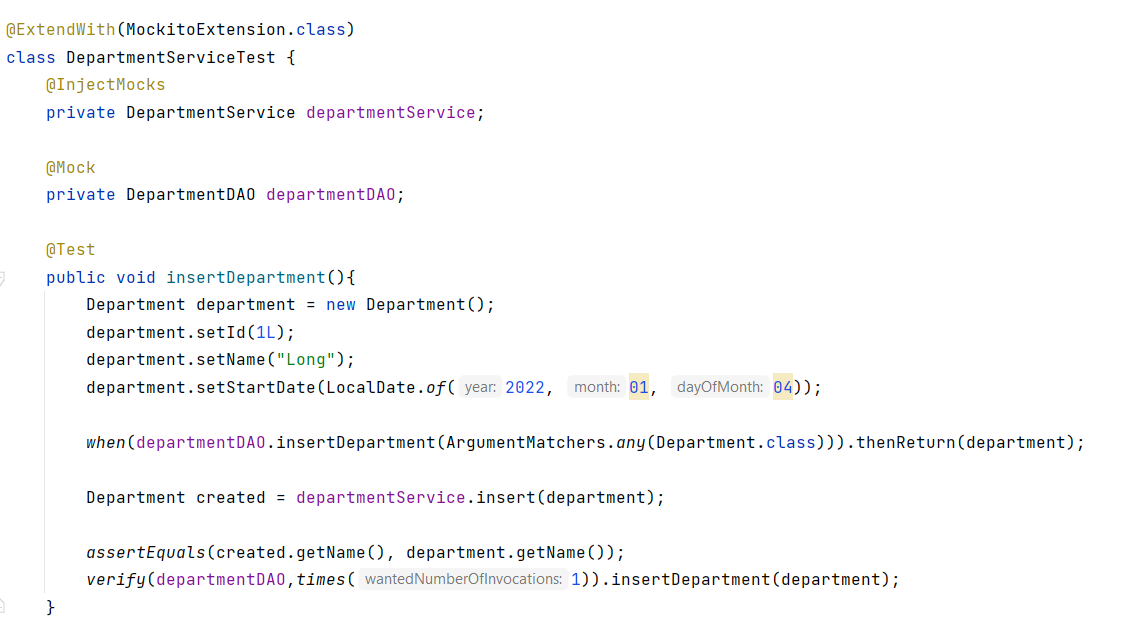




and a service class:



So, we will test this service:



Now, let’s analyze the code:

* @Mock: create a mock object of DepartmentDAO
* @InjectMocks creates an instance of the class DepartmentService and injects the mocks that are created with the @Mock annotations into this instance. (inject DepartmentDAO into DepartmentService)
* Then I create a Department.
* when() and thenReturn() method I have mentioned above. But in this case we have ArgumentMatchers.any(Department.class). It mean, we don’t know which object will insert with method insertDepartment(), so ArgumentMatchers.any() allow us declare it, any Department.
* Then run method insert of DepartmentService and compare with department name.
* verify(): Mockito can ensure whether a mock method is being called with required arguments or

not. Here I check that the method insertDepartment() of departmentDAO is called or not

* You can also verify number of invocation times on the method with times().

I just create simple Mockito test method for a service class, Mockito has a lot of annotation and method (thenThrow(), thenAnswer()…). I will research about REST API testing with Mockito and update later

**Fix run test on Java EE (sometimes):**

**In intellij, setting -> search "maven" -> repositories -> click to type : local -> update to fix error surefire-plugin:2.19 not found in POM file.**

You also can research more about it through some documents:

<https://www.tutorialspoint.com/mockito/index.htm>

<https://javadoc.io/doc/org.mockito/mockito-core/5.2.0/index.html>

https://semaphoreci.com/community/tutorials/stubbing-and-mocking-with-mockito-2-and-junit

https://javadoc.io/doc/org.mockito/mockito-core/latest/org/mockito/Mockito.html