# **Mockito & Power Mock**

Mockito:

Mockito is a mocking framework, JAVA-based library that is used for effective unit testing of JAVA applications. Mockito is used to mock interfaces so that a dummy functionality can be added to a mock interface that can be used in unit testing.

1. @Mock

It is used to mock the objects that helps in minimizing the repetitive mock objects. It makes the test code and verification error easier to read as parameter names (field names) are used to identify the mocks. The @Mock annotation is available in the **org.mockito** package.

Ex:

;

1. @Inject Mock

It marks a field or parameter on which the injection should be performed. It allows shorthand mock and spy injections and minimizes the repetitive mocks and spy injection. In Mockito, the mocks are injected either by setter injection, constructor injection, and property injection. The @InjectMocks annotation is available in the **org.mockito** package.

Ex:



1. @RunWith

It is a class-level annotation. It is used to keep the test clean and improves debugging. It also detects the unused stubs available in the test and initialize mocks annotated with @Mock annotation. The @RunWith annotation is available in the **org.mockito. junit** package.

Ex:



1. @Captor

It allows the creation of a field-level argument captor. It is used with the Mockito's verify () method to get the values passed when a method is called. Like other annotations, @Captor annotation is also available in the **org. mockito**package.

Ex:



1. Mockito Junit Rule

In the above examples, we have used the JUnit **runner (MockitoJUnitRunner).** It makes the test dependent on that particular runner. We cannot use multiple runners in the same test. To overcome this problem, we should follow **JUnit rules** that makes the test more flexible. It allows us to use multiple rules in the same test.

Ex:

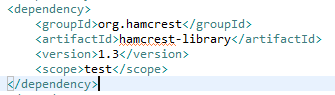


## Ham crest:

**Ham crest** is a popular framework that help us to create the matcher objects. It is used for writing software tests and performs unit testing in Java programming language. Ham crest is mainly used with other unit testing frameworks like **JUnit, j Mockito, Mockito,** etc.

The Ham crest framework was designed to accommodate different types of unit testing frameworks. For example, Ham crest can be used with TestNG and JUnit (all versions). The Ham crest framework is also used with mocking frameworks such as J Mock, Easy Mock, and Mockito.

Add Dependency**:**



1. Mockito to Spy

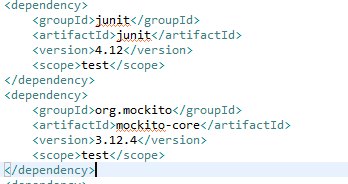
It allows the creation of partially mock objects. In other words, it allows shorthand wrapping of the field instances in a spy object. Like other annotations, @Spy annotation is also available in the **org. mockito** package.

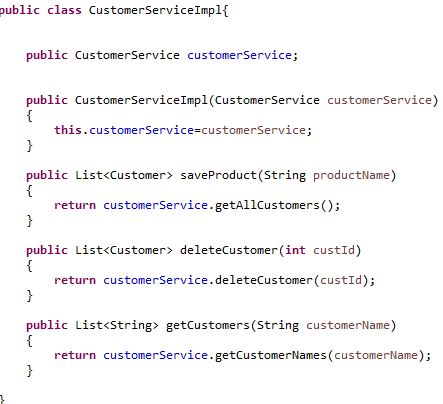
Ex:

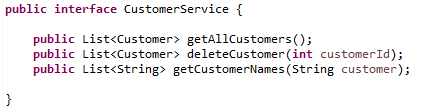
@Spy

ArrayList<String> arraylistSpy;

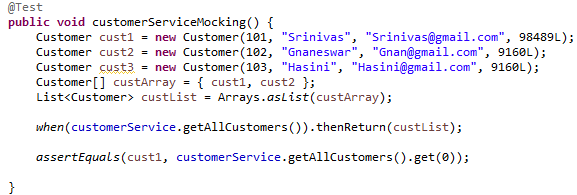
1. Sample Mockito Example:
2. Create Sample Maven project.
3. Add below dependencies to work with Mockito.



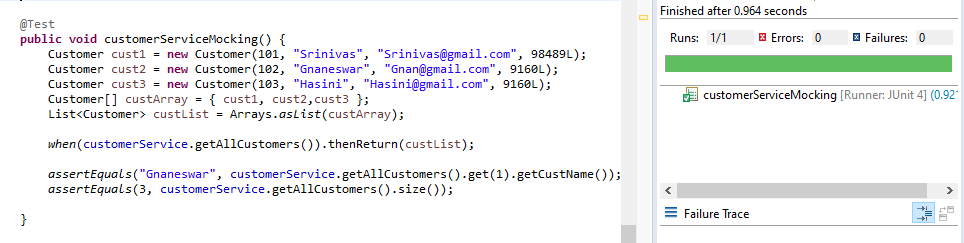
1. Create Service & Service implementation which is having actual business logic.



1. Create Sample Test class in src/test/java and create multiple test cases.



1. Run as Junit Test



**POWER MOCK**

What is POWERMOCK:

Power Mock is an open-source Java framework used for creating a mock object in unit testing. It extends other mocking frameworks such as Easy Mock and Mockito to enhance the capabilities.

Why POWERMOCK:

The main aim of Power Mock is to extend the existing APIs with some methods and annotations to provide extra features that make unit testing quite easy.

User of POWERMOCK:

The Power Mock framework uses a custom class loader and bytecode manipulation techniques to enable the mocking of static methods, final classes, final methods, private methods, constructor, and removal of static initializers.

To use Power Mock with Mockito, we need to apply the following **two** annotations in the test:

@RunWith (PowerMockRunner.class):

 It is the same as we have used in our previous examples. The only difference is that in the previous example we have used MockitoUnitRunner.class, now we will use PowerMockRunner.class for enabling the Power Mockito APIs in the test.

@PrepareForTest:

It tells Power Mock to prepare some classes for testing. It can be applied to both the test classes and the individual test methods. It includes classes with final, static, private, or native methods that can be mocked.

Sample Power Mock Example:

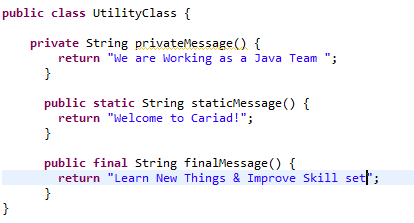
1. Create Sample Maven project.
2. Add below dependencies.

Dependency:

The following dependency need to add to use Power Mock in the application.



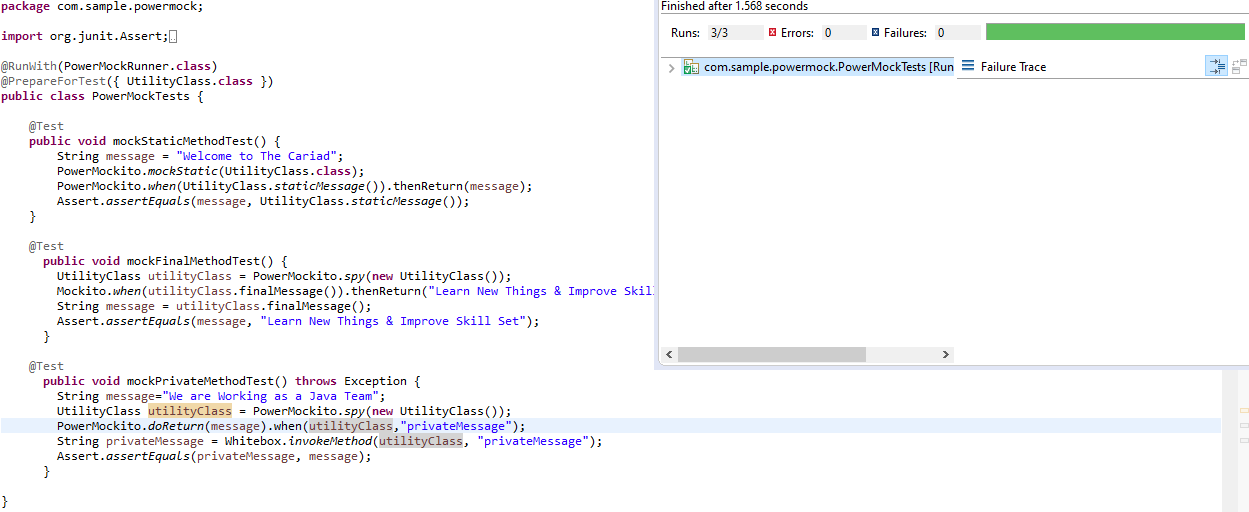
1. Create Sample Class and create static, private & final functions.



1. Implement test cases for Utility Class by using Power Mock.



1. Run as Junit Test



**Thanks!**