Mockito Basics

Mockito

DEFINITION/WHAT

Open source testing framework for Java, It is a java based mocking framework.

The purpose of mocking is to isolate and focus on the code being tested and not on the behavior or state of external dependencies

NEED/WHY

Need 1 : Used for effective unit testing of JAVA applications

Need 2 : Mockito is used to test the functionalities of the classes without depending on external dependencies as database connection or properties file read or file server read to test a functionality

Need 3 :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

Need 4: No need to write mock objects on your own with Mockito

IMPLEMENTATION/HOW IT INTERNALLY WORKS

It internally uses Java Reflection API and allows to create objects of a service

Mock objects do the mocking of the real service.

A mock object returns a dummy data corresponding to some dummy input passed to it

With Mockito, you create a mock, tell Mockito what to do when specific methods are called on it, and then use the mock instance in your test instead of the real thing

After the test, you can query the mock to see what specific methods were called or check the side effects in the form of changed state

REAL TIME EXAMPLE

```
@Test
void testCalc() {
        AddService addService;
        CalcService calcService;

        addService = Mockito.mock(AddService.class);
        calcService = new CalcService(addService);

        int num1 = 11;
```

```
        int num2 = 12;
        int expected = 23;

        when(addService.add(num1, num2)).thenReturn(expected);
        int actual = calcService.calc(num1, num2);
        assertEquals(expected, actual);

    }
```

ALTERNATIVE/DIFFERENCES

How mockito is different than junit

Junit:-JUnit is a framework that helps with writing and running your unit tests

Mockito:-Mockito is a library that enables writing tests using the mocking approach.

ADDITIONAL INFORMATION

There are broadly 2 categories of mocking frameworks:

Proxy-based – Example, Mockito, EasyMock, etc.

Bytecode based – Example, PowerMock, JMockit, etc.

Mockito Methods

mock() method

Definition/What

Used to create mock objects of a given class or interface

Implementation/How it internally works

Mockito contains five mock() methods with different arguments, arguments like class or answer or MockSettings or Returnvalues or string

Real Time example

ToDoService doService = mock(ToDoService.class);

Alternative/Differences

The other way of creating the mock instances is using the @Mock annotations.

when() method

Definition/What

It enables stubbing methods

Need/Why

Need 1 : Used when we want to mock to return specific values when particular methods are called.

Need 2: It is mostly used when there is some condition to execute.

Implementation/How it internally works

It is written as <T> when(T methodCall)

Real Time Example

when(mock.someCode ()).thenReturn(5);

Here, thenReturn() is mostly used with the when() method.

verify() method

Definition/What

It validates the certain behavior that happened once in a test

Need/Why

Need 1 : Used to check whether some specified methods are called or not.

Need 2: Used in behavioral testing, It checks that a method is called with the right parameters instead of checking the result of a method call.

Need 3: Used to test the number of invocations, (using times method, at least once method, and at most method) for a mocked method.

Implementation/How it internally works

It is used at the bottom of the testing code to assure that the defined methods are called.

It is written as <T> verify(T mock)

Real Time Example

```
@Test
void test() {
	List<String> mockList = mock(List.class);
	mockList.add("ABC");
	mockList.size();
	verify(mockList).add("ABC");
}
```

Above verify method will pass if add("ABC") is called only once on the mocked list object

thenReturn() method

Definition/What

Lets you define the return value when a particular method of the mocked object is been called.

Need/Why

Need 1 : You can specify what to return when a method is called, which makes unit testing easier because you don't have to change existing classes.

Need 2: If we need fixed return value on method call then we should use thenReturn(…)

Implementation/How it internally works

Method thenReturn() needs a fixed object which will be returned when we call the method.

We can pass any type of object or value, the same value will be returned on method call

I.e. thenRetun() will always return the same object.

It is written as: OngoingStubbing<T> thenReturn(T value)

Real Time Example

Ex1:

```
Iterator i = mock(Iterator.class);
when(i.next()).thenReturn("Mocking Framework").thenReturn("Mockito");
String result = i.next() + " " + i.next();
System.out.println(result);
```

Here, The first time next() method is called 'Mocking Framework' is returned and when it's called the second time 'Mockito' is returned. So the result is Mocking Framework Mockito

Ex:2

```
when(mockEmployeeDAO.deleteEmployee(1001L)).thenReturn("DELETED");
```

Additional Information

The thenReturn way of stubbing is a type-safe way of setting up stubs

What this essentially means is that it does a compile-time check against the return types that you want to stub too

Mockito Annotations

@Mock

Definition/What

Used to create and inject mocked instances

The @Mock annotation is alternative to Mockito.mock(classToMock)

Need/Why

Need 1 : Use @Mock to create and inject mocked instances without having to call Mockito.mock manually, allows shorthand creation of objects required for testing.

Need 2: Minimizes repetitive mock creation code.

Need 3: Makes the test class more readable.

Real Time Example

```
@Mock
List<String> mockedList;

@Test
public void whenUseMockAnnotation_thenMockIsInjected() {
   mockedList.add("one");
   Mockito.verify(mockedList).add("one");
    assertEquals(0, mockedList.size());

   Mockito.when(mockedList.size()).thenReturn(100);
   assertEquals(100, mockedList.size());
}
```

@InjectMocks

Definition/What

allow us to inject mocked dependencies in the annotated class mocked object.

Need/Why

Need 1 : Used to create class instances which needs to be tested in test class

Need 2: Used when actual method body needs to be executed for a given class

Need 3: Used when we need all internal dependencies initialized with mock objects to work method correctly

Implementation/How it internally works

Mockito tries to inject mocked dependencies using one of the three approaches- Constructor Based Injection or Setter Methods Based or Field Based

Real Time Example

```
@InjectMocks
AppServices appServicesConstructorInjectionMock;

@Test
void test_constructor_injection_mock() {

when(appServicesConstructorInjectionMock.sendEmail("Email")).thenReturn(true);
when(appServicesConstructorInjectionMock.sendSMS(anyString())).thenReturn(true)

assertTrue(appServicesConstructorInjectionMock.sendEmail("Email"));
assertFalse(appServicesConstructorInjectionMock.sendEmail("Unstubbed Email"));

assertTrue(appServicesConstructorInjectionMock.sendSMS("SMS"));

}
```

Alternative/Differences

@Mock Vs @InjectMocks

@Mock : creates a mock

@InjectMocks : creates objects and inject mocked dependencies.