Differences between Stubbing and Mocking

Category: Blogs , August 6 2021

0

0

7 min read

Author

Ravinder Singh's Photo

Ravinder Singh

Reviewers

Virender Singh's Photo

Virender Singh

Stubbing and Mocking are two fundamental approaches in the testing field. However, they are frequently misapprehended. There is a common perception that stubbing and mocking happen only for unit and component testing. But we can use them for other testing types as well. Subsequently, let us explore the differences between stub and mock by discussing the below topics:

What is a stub?

What is mock?

Also, what are the differences between stub and mock?

When to use a stub?

When to use a mock?

What is a stub?

The stub is an interface that has a minimum count of methods to simulate the actual object. It is an object that has preexisting data and returns a fixed value irrespective of input. Moreover, we mainly use it when we want to avoid a response from the real object.

Primarily, we use it wherever the test suite is uncomplicated, and having hard-coded values is not a problem. Additionally, both developers and testers use it. But we can not share it mainly for compatibility concerns occurring due to hard-coding of resources, deployment dependencies, and platforms.

We can also use the stub for communication over HTTP, which some name as virtual service. Additionally, we can also use it to match database objects. In short, a stub can be used as a dummy module to verify an application when the actual module is still under implementation. Moreover, the stub eliminates the concerns that may come up while implementing the actual object.

Subsequently, let us take an example of a game under development with Play, Exit, Up, and Down Arrow buttons.

When game development is in progress, we have to verify user interfaces (Play, Exit, Up, and Down arrows), we shall implement some dummy actions. For example, on clicking on the Play button, a dummy action shall take place. Additionally, we can do the implementations as shown below:

//Play button stub method

public void playbtn(){

System.out.println("Play button working fine");

}

//Exit button stub method

public void exitbtn(){

System.out.println("Exit button working fine");

}

//Up Arrow stub method

public void upArw(){

System.out.println("Up arrow working fine");

}

//Down Arrow stub method

public void downArw(){

System.out.println("Down arrow working fine");

}

Next, to implement the above actions as a stub, we will include the logic as depicted below:

if(button = "Play"){

playbtn();

}

if(button = "Exit"){

exitbtn();

}

if(button = "Uparrow"){

upArw();

}

if(button = "Downarrow"){

downArw();

}

What is mock?

The mock is an interface that we program to check the outputs that are results of the tests against the expected results. We commonly implement it by taking the help of third-party libraries like Mockito, JMock. It is also helpful when we have a big test suite, and each test requires a unique set up of data.

In such a scenario, maintaining a stub becomes an expensive affair. A mock allows maintaining data configuration within the test. Moreover, both the developers and testers can use it. But we can not share it with a broader group due to compatibility concerns occurring from hard-coding of resources, deployment dependencies, and platforms.

Additionally, a mock can also count how many times we call a method, function, etc. Or the sequence of calls to an object. Moreover, it checks the communications among classes. We can use the method mock() for mocking purposes.

Subsequently, let us see the implementation for testing of mock objects via the Mockito framework.

//interface

public interface Mathematics{

public int sub(int a, int b){

}

}

// class

public class Substraction{

//getter method

public Mathematics getSub(){

return m;

}

//setter method

public setSub(Mathematics m){

this.m = m;

}

Mathematics m;

public int subNumbers(int a, int b){

return m.sub(a,b);

}

}

The below example shows an implementation of a mock object for interface Mathematics within a Mockito test.

import org.junit.Before;

import org.junit.Test;

import static org.mockito.Mockito.\*;

public class MathematicsTest{

Subtraction s;

@Before

public void setup(){

// mock object created for Interface Mathermatics

Mathematics maths = mock(Mathematics.class);

when(maths.sub(2,1)).thenReturn(1);

s = new Substraction();

s.setSub(maths);

}

@Test

public void testsubNumbers(){

Assert.assertEquals(1, s.subNumbers(2,1));

}

}

What is the difference between stub and mock?

Conclusively, now that we have a basic understanding of stub and mock, we shall list their differences based on some features.

Features Stub Mock

Implementation Implemented by developers, testers by themselves, or produced via tools. The developers implement it via a third-party library like Mockito, JMock, and WireMock.

Scenario Uncomplicated test suites use it. Big test suites use it.

User Interface No graphical user interface. It has a graphical user interface.

Data Setup Data gets hard-coded. Here, data configuration happens by the tests.

Purpose Its purpose is state verification. Its purpose is characteristics verification.

Coupling of Data It tightly couples with test data. Workable with both tightly and loosely coupled test data.

Stateful No Yes

Advantages Here, Free tools are available, along with plenty of resources online. Here, Open-source tools are available, along with plenty of resources online.

Disadvantages Test cases couple tightly due to the hard-coding of data. Mostly used by developers and not by testers.

Usage We use it if the test data requirements in the suite are simple. We use it if the test data requirements in the suite are complex also if all the data configurations happen within the tests.

Technical Knowledge It requires average technical knowledge. It requires significant technical knowledge.

When to use a stub?

We can implement stub in the scenarios listed below:

If we are developing the back end of a minor application with a third-party library to interact with an API, we can implement an interface that can disconnect us from the third-party library. Eventually, that interface will act as a stub and yield hard-coded values. Conclusively, we can use values in unit tests.

If we are testing an independent application, we will implement a stub for the Rest APIs on which the application is built. Additionally, we can create it with the help of any internal tool designed by the development team.

When to use a mock?

We can implement a mock object in the scenarios listed below:

If we are developing a back end of an application having many classes to test, we can use a Mockito framework to mock the dependent classes.

If we are developing a back end of an application and we have to disconnect from the API dependencies in HTTP, we can use a mock framework like mountebank or WireMock. Then create a mock for the dependency classes in the test.

Key Takeaways

A stub is an object which has preexisting data and utilizes it during tests. Moreover, we mainly implement when we avoid actual objects interacting with data. Additionally, it produces an unacceptable outcome.

A mock object is similar to a stub. However, it is possible to apply assertions on a mock object which is not there for a stub object.