Junit & Mockito :

What is JUnit?

JUnit is a unit testing framework for Java programming language. It provides annotations to identify methods that specify a test, and assertions to check expected results.

What are the advantages of using JUnit for testing?

JUnit provides a standardized way to write and execute test cases.

It facilitates automated testing, reducing manual testing efforts.

It encourages developers to write testable code by following the principles of test-driven development (TDD).

It allows for easy integration with build tools like Maven and Gradle.

Explain the basic JUnit annotations.

@Test: Indicates that a method is a test method.

@Before: Executed before each test method to set up preconditions.

@After: Executed after each test method to clean up resources.

@BeforeClass: Executed once before any test methods in the test class.

@AfterClass: Executed once after all test methods in the test class.

What is the purpose of assertions in JUnit?

Assertions are used to verify that expected results match actual results in a test case. They help ensure that the code behaves as expected.

assertTrue(boolean condition):

This method verifies that the given condition is true. If the condition is false, the test fails.

assertTrue(result > 0);

assertFalse(boolean condition):

This method verifies that the given condition is false. If the condition is true, the test fails.

assertFalse(errors.isEmpty());

assertNull(Object object):

This method verifies that the given object reference is null. If the object is not null, the test fails.

assertNotNull(Object object):

This method verifies that the given object reference is not null. If the object is null, the test fails.

Explain parameterized tests in JUnit.

Parameterized tests allow you to run the same test method multiple times with different sets of parameters. This is useful when you want to test a method with different inputs.

CsvSource  
Passing record separated by comma,each record is unique  
Valuesource  
Passing an array separated by comma or single number as a source

What are the differences between JUnit 4 and JUnit 5?

JUnit 5 introduced several new features and improvements over JUnit 4, including support for parameterized tests, nested tests, conditional test execution, and extensions. Additionally, JUnit 5 uses annotations from the org.junit.jupiter.api package instead of org.junit.Test for test methods.

Testing exceptions in JUnit allows you to verify that certain code throws expected exceptions under specific conditions. There are several ways to test exceptions in JUnit, depending on the version of JUnit you're using.

Here's how you can test exceptions in JUnit:

JUnit 4:

Using @Test annotation's expected attribute:

You can specify the expected exception class using the expected attribute of the @Test annotation.

java

Copy code

import org.junit.Test;

public class MyTestClass {

@Test(expected = ArithmeticException.class)

public void testDivideByZero() {

int result = 10 / 0;

}

}

Using ExpectedException rule:

You can use JUnit's ExpectedException rule to specify the expected exception and any additional assertions related to the exception.

java

Copy code

import org.junit.Rule;

import org.junit.Test;

import org.junit.rules.ExpectedException;

public class MyTestClass {

@Rule

public ExpectedException exceptionRule = ExpectedException.none();

@Test

public void testDivideByZero() {

exceptionRule.expect(ArithmeticException.class);

exceptionRule.expectMessage("/ by zero");

int result = 10 / 0;

}

}

JUnit 5:

Using assertThrows() method:

JUnit 5 provides the assertThrows() method in the Assertions class to assert that a particular piece of code throws an exception.

java

Copy code

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertThrows;

public class MyTestClass {

@Test

public void testDivideByZero() {

assertThrows(ArithmeticException.class, () -> {

int result = 10 / 0;

});

}

}

Mockito Questions:

What is Mockito?

Mockito is a mocking framework for Java that allows you to create and configure mock objects. Mock objects simulate the behavior of real objects in controlled ways during tests.

Explain the concept of mocking in Mockito.

Mocking involves creating fake objects that simulate the behavior of real objects in a controlled manner during tests. Mockito allows you to define mock objects and specify their behavior using stubbing.

What are the advantages of using Mockito?

Mockito simplifies the testing of classes with dependencies by allowing you to mock those dependencies.

It provides fluent APIs for defining mock behaviors and verifying interactions with mock objects.

Mockito promotes test code readability and maintainability.

Explain the difference between mock and spy in Mockito.

A mock object is created entirely by Mockito and has no real implementation. You define its behavior using Mockito's API.

A spy object is a real object that is spied on by Mockito. It retains its original behavior unless stubbed otherwise.

How do you verify interactions in Mockito?

Mockito provides methods like verify() to verify that specific interactions with mock objects occurred during the test execution.

What is stubbing in Mockito?

Stubbing refers to defining the behavior of a mock object. You specify the return value of methods or throw exceptions when they are called on mock objects.

What are Mockito annotations, and how are they used?

Mockito annotations like @Mock, @InjectMocks, and @Spy are used to simplify the creation and injection of mock objects in tests. @Mock is used to create mock objects, @InjectMocks is used to inject mocks into the test class under test, and @Spy is used to spy on real objects.

What are the differences between stubbing and mocking in Mockito?

Stubbing involves defining the behavior of mock objects by specifying return values or throwing exceptions when certain methods are called. Mocking, on the other hand, refers to the creation of mock objects themselves.

8. What is a junit fixture

Answer: In JUnit, a fixture refers to the preparation and setup needed for a test case to run.

It includes creating the necessary objects, initializing variables, and configuring the

environment before executing the actual test. The @Before annotation in JUnit is

commonly used to define fixture setup methods, ensuring consistency and repeatability

in test execution.

7. What is a test suite?

Answer: A test suite is a collection or grouping of individual test cases that are executed

together. It allows developers to organize and run multiple tests as a single unit, making

it easier to manage and analyze the results of various test scenarios. Test suites are

commonly used in testing frameworks like JUnit to streamline the testing process.

6. What are annotations and how are they useful in JUnit?

Answer: Annotations are like meta-tags that you can add to you code and apply them to

methods or in class. The annotation in JUnit gives us information about test methods ,

which methods are going to run before & after test methods, which methods run before

& after all the methods, which methods or class will be ignore during execution.

1) Give some disadvantages of manual testing.

Following are some disadvantages of manual testing:

The testing is very time consuming and is very tiring.

The testing demands a very big investment in the human resources.

The testing is less reliable

The testing cannot be programmed.

4) List out some advantages of automated testing.

Some of the advantages of automated testing are:

It is very fast.

Investment is very less.

Testing is more reliable.

The testing can be programmed.

5) Is it necessary to write the test case for every logic?

No, we should write the test case only for that logic that can be reasonably broken.

6) What are the useful JUnit extensions?

JWebUnit

XMLUnit

Cactus

MockObject

7) What are the features of JUnit?

Opensource

Annotation support for test cases

Assertion support for checking the expected result

Test runner support to execute the test case

8) How is the testing of the ‘protected’ method done?

To test the protected method, the test class is declared in the same package as the target class.

9) How is the testing of ‘private’ method done?

There is no direct way for testing of the private method; hence manual testing is to be performed, or the method is changed to “protected” method.

10) If the JUnit method’s return type is ‘string’, what will happen?

JUnit test methods are designed to return ‘void’. So the execution will fail.

11) Is the use of ‘main’ method possible for unit testing?

Yes

12) Is it necessary to write the test class to test every class?

No

13) What does XMLUnit provide?

Junit extension class, XMLTestCase and set of supporting classes is provided by the XMLUnit.

14) List the core components of Cactus.

Cactus Framework

Cactus Integration Module

15) What are the methods in fixtures?

setup

tearDown

16) What is the Unit Test Case?

A Unit Test Case is the combination of input data and expected output result. It is defined to test the functionality of a unit.

17) What is the use of @Test annotation?

The @Test annotation is used to mark the method as the test method.

18) What is the test suit?

The test suit allows us to group multiple test cases so that it can be run together. TestSuit is the container class under junit.framework.TestSuite package.

19) What does test runner?

The test runner is used to execute the test cases.

20) What are the important JUnit annotations?

The test runner is used to execute the test cases.

@Test

@BeforeClass

@Before

@After

@AfterClass

21) What is JUnit?

JUnit is a unit testing framework for the Java application. It allows you to unit test your code at the method level. You can also use JUnit for test-driven development e.g. first writing test and then writing actual code. Most of the Java IDE like Eclipse, Netbeans and IntelliJ Idea provides out-of-box JUnit integration for TDD. Even build tools like Maven, Ant, and Gradle can run the JUnit test at compile and built time.

22) What is the difference between JUnit 3.X and JUnit 4.x?

The main difference between JUnit 3 and JUnit 4 is annotation. Earlier you need to start your method name as testXXX() but from JUnit 4.0 onwards you can use @Test annotation, which removes this dependency to start your Junit test methods as a test.

23) What do @Before and @After annotation does in JUnit 4?

@Before and @After annotation is used for setup and cleanup jobs. They are actually a replacement of the setUp() and tearDown() method of JUnit 3.X. Remember, a method annotated with @Before will be called before calling a test method and @After will be called after executing the test method.

So if you have 5 test methods in one JUnit test class, they will be called five times. You will be glad to know that JUnit 5 has renamed @Before to @After to @BeforeEach and @AfterEach to make it more readable.

24) How does JUnit achieve the isolation of tests?

JUnit creates a separate instance of the test class for calling test methods. For example, if your test class contains 5 tets then JUnit will create 5 instances of the test class for executing those test methods. That’s why constructor is called before executing @Test methods. In JUnit 3.X, It used to first create instances and then call the methods but in JUnit 4.1, it first creates an instance and then calls

ADVERTISEMENT them the method and then goes on to create the second method. This way, it keeps one test isolated from the other.

25) What is difference between setUp() and tearDown() method?

They are methods from JUnit 3.X which is replaced by @Before and @After annotated method. They are used for setting up resources before the test method and for doing clean-up after-test methods.

26) How do you check if a test method throws Exception in JUnit?

In JUnit 4, you can use @Test annotation with an expected parameter to specify the exception thrown by a test method. If the test method throws that Exception then JUnit will mark your test passed, otherwise, it will mark them as failed.

For example, if you want to verify your method should throw IllegalArgumentException if the given parameter is bad then you can annotate your test method as @Test(expected = IllegalArgumentException.class). In earlier versions e.g. on JUnit 3.X, you could have used catch block to verify if the method under test throws a particular exception or not.

27) How do you ignore certain test methods in the JUnit Test class?

Well, from JUnit 4 onwards you can use @Ignore annotation to instruct JUnit runtime to not execute any particular test or disable any test as shown in this JUnit ignore example. You can also use this method for annotating unimplemented test methods. Btw, you should use @Ignore only temporarily and remove it as soon as possible by either implementing the test methods or removing it if not required.

28) What are some best practices you follow while writing code to make it more testable?

Here are some of the best practices you can follow to make your code more testable :

1) Use interface instead of concrete class, this allows the tester to replace actual class with Mock or Stub for testing purposes.

2) Use Dependency injection, it makes it easy to test individual parts and supply dependency from test configuration. You can create an actual object which is dependent on the Mock and stub for testing. This also allows you to test a class even if its dependency is not yet coded.

3) Avoid static methods, they are difficult to test as they cannot be called polymorhpically.

29) What does @RunWith annotation do?

The @RunWith annotation allows you to run your JUnit test with another, custom JUnit Runner. For example, you can do some parameterized testing in JUnit by using @RunWith(Parameterized. class), this Runner is required for implementing parameterized tests in JUnit.

Some of the popular third-party implementations of JUnit runners include SpringJUnit4ClassRunner and MockitoJUnitRunner, which you can use with @RunWith annotation to test Spring beans and Mockito library.

30) What happens when your JUnit test method throws an exception?

When a JUnit method throws an uncaught exception, which is also not declared as expected using the @Test(expected) parameter then the JUnit test will fail, but if it’s declared and expected then the JUnit test will pass. If the method under test throws an exception that is caught by you inside the JUnit test method then nothing will happen and the test will be passed

31) How do you test protected Java methods using JUnit?

Since protected methods are accessible to everyone inside the same package they are declared, you can put your JUnit test class also on the same package to test them. But, don’t mix your test code with real code, instead, put them on a different source tree. You can use a Maven-like structure to organize your test and source classes in different folders.

32) Can you test a private method using JUnit?

Well, since private methods are not accessible outside the class they are declared, it’s not possible to test them using JUnit directly, but you can use Reflection to make them accessible by calling setAccessible(true) before testing. Another way to test private methods is via public methods, which uses them. In general, testing the private method is also an indication that those methods should be moved into another class to promote reusability.

33) What are some Unit testing libraries you know other than JUnit for Java programs?

Java is very lucky to have many great unit testing libraries including JUnit. Here are a couple of the most popular ones:

Mockito which is a Mocking framework can be used with JUnit

PowerMock, similar to Mockito but also provide mocking of static methods.

TestNG is similar to JUnit

Spock groovy-based unit testing framework for the Java application. Thanks to JUnit Runner, you can use the Spock testing framework with Java IDEs as well.

At a bare minimum, you should JUnit and Mockito for writing unit tests in Java.

34) What does @Rule annotation do?

It allows you to create JUnit rules, which can be used to add or redefine the behavior of each test method in a test class. You can write your own rules using @Rule annotation.

35) What are text fixtures in unit testing?

A test fixture in unit testing is used to provide a well-known and fixed environment in which the test can be run. A text fixture could be set up or mock objects or loading specific data from the database for testing. JUnit provides @Before and @BeforeClass annotation for creating text fixtures and @After and @Afterclass for doing cleanup.

36) Define JUnit in today’s context of evolving technologies?

JUnit is also known as a waning testing framework. JUnit is extensively used by developers to carry out unit testing in Java. Moreover, it is also being used to speed up the application based on Java. It is important to note that by taking into account the source code, the application can efficiently be sped up.

37) When are the unit tests pertaining to JUnit written in Developmental Cycle?

The unit tests are written before the development of the application. It is so because writing the check before coding, assists the coders to write error-free codes which further boosts the viability of the form.

39) What are the critical fixtures of JUnit?

The JUnit test framework is associated with the providence of these critical features. They are as follows:

Test Suites

JUnit Classes

Fixtures

Test Runners

40) Define installation with respect to JUnit?

A fixture is also known as a constant state of a collection of objects that can be used for the execution of tests. The main aim of using a test fixture lies in the fact that there should be a familiar and fixed environment. Moreover, these tests are run so that the results are repeatable. It comprises of the following procedures.

The setup method which runs every test

The teardown procedure which runs after the execution of a particular test

41) Give a brief account of the Test Suite with respect to JUnit?

Test Suite usually refers to the principle of compiling a variety of unit test cases to run it concurrently. In this context, it is interesting to note that in JUnit, both Run With and Suite comments are being used to avail maximum benefits.

42) What is a test fixture with respect to JUnit?

A test fixture is also known as a regulated state of objects that can be used as a platform for running the tests. The primary purpose is to make sure that there is a known climate in which the development tests can be run. Various examples can be cited in this context. They are as follows:

Copying the fixed known set of files

Preparing the input data and the creation of mock and fake objects

Assessing a database with fixed and known sets of data

It is also essential for you to note that if a group of tests shares the same fittings, one needs to write a different setup code. On the other hand, if the group of assessments is in need of a different test fixture, one can write the code alongside the test procedure. In this manner, one can create the best accessory related to a test.

43) Shed light on the reason that why JUnit only reports the first failure in a single attempt?

It is interesting to note that saying multiple shortcomings in one attempt signifies that the test is of a larger size. This is the reason that JUnit is designed in such a manner that it usually runs on a smaller number of tests. Quite interestingly, it is capable of executing each assessment within the boundaries of a separate analysis. Moreover, it is also able to report the failures on each attempt of the tests.

44) Shed your views on the @ignore annotation. Even describe its usefulness?

Here is the list of advantages of using this kind of annotation

It is important to note that one can easily find @ignore annotations with relatively much ease that lies inside the source code. On the other hand, the unannotated and uncommented tests are complicated to find.

Moreover, one may come across various situations where one cannot fix a code. However, if you still want that source method to be taken into account, @ignore annotation plays a crucial role in that.

45) What do you mean by parameterized tests in JUnit?

In version 4 of JUnit, there is a new feature known as Parameterized Tests. These tests usually allow the developer to run the same amount of tests over and again. In this manner, the suitability of using various values increases considerably.

46) Describe the process which is helpful in the creation of parameterized Tests?

In order to answer this question, you need to take into account that there are five steps. These steps are as follows:

The first step is that you have to create a public constructor which would take into account the equivalent value of a row of test-related data

In the second step, you have to annotate the test class with a command by the name of @Run With

In the third step, you have to create a public static procedure that has the ability to return a collection of test data sets

In the fourth step, you need to create a viable which is instantaneous and is the source of data

In the last level, you have to make sure that the test case is invoked only once per row. This would facilitate the smooth running of an application.

47) What is a Unit Test Case?

A Unit Test Case is a part of code which ensures that the another part of code (method) works as expected. A formal written unit test case is characterized by a known input and by an expected output, which is worked out before the test is executed. The known input should test a precondition and the expected output should test a post condition.

48) Why does JUnit only report the first failure in a single test?

Reporting multiple failures in a single test is generally a sign that the test does too much and it is too big a unit test. JUnit is designed to work best with a number of small tests. It executes each test within a separate instance of the test class. It reports failure on each test.

49) In Java, assert is a keyword. Won’t this conflict with JUnit’sassert() method?

JUnit 3.7 deprecated assert() and replaced it with assertTrue(), which works exactly the same way. JUnit 4 is compatible with the assert keyword. If you run with the -ea JVM switch, assertions that fail will be reported by JUnit.

50) When are the unit tests pertaining to JUnit written in Developmental Cycle?

The unit tests are written before the development of the application. It is so because by writing the check before coding, it assists the coders to write error-free codes which further boosts the viability of the form.

51) What are the different methods of exception handling in JUnit?

There are different ways of exception handling in JUnit. Try catch idiom.

Using JUnit rule.

@Test annotation.

Using catch exception library.

Using customs annotation.

52) How do I execute JUnit from command line?

Executng Junit from command line involves 2 steps. set the ClassPath to include JUnit core libraries. set CLASSPATH=%CLASSPATH%;%JUNIT\_HOME%junit.jar

2.Invoke JunitCore. java org.junit.runner.JUnitCore

53) How do I test a private method?

The private methods cannot be tested as any private method only be accessed within the same class. The private method need to tested manually or be converted to protected method.

54) What happens if a test method throws an exception?

The JUnit runner will declare that test as fail when the test method throws an exception.

The methods Get () and Set () should be tested for which conditions?

Unit tests performed on java code should be designed to target areas that might break. Since the set() and get() methods on simple data types are unlikely to break, there is no need to test them explicitly. On the other hand, set() and get() methods on complex data types are vulnerable to break. So they should be tested.

55) What is JUnit parameterized test?

The parameterized test is a new feature introduced in JUnit 4. It provides the facility to execute the same test case again and again with different values.

56) Name the tools with which JUnit can be easily integrated.

JUnit Framework can be easily integrated with either of the followings − Eclipse

Ant

Maven.

57) Difference between Assert and Verify ?

Assert works only if assertions ( -ea ) are enabled which is not required for Verify. Assert throws an exception and hence doesn’t continue with the test if assert evaluates to false whereas it’s not so with Verify.

58) Place the JUnit program in the context of the current evolution of tech and programming.

Junit is known as a waning testing framework. It is extensively utilized by developers to carry out unit testing in Java. At the same time, it is also commonly used to speed up applications based in Java. It is crucial to note that, through a consideration of the source code, the application would be efficiently sped up.