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.

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.