1) what is JUnit?

JUnit is a testing framework for unit testing. It uses Java as a programming platform, and it is an Open Source Software managed by the JUnit.org community.

2) what is Unit Test Case?

Unit Test Case is a part of the code that ensures that the another part of code (method) behaves as expected. For each requirement, there must be at least two test cases one negative test and one positive test.

3) To write a simple JUnit test case?

Determine a subclass of TestCase

To initialize object(s) under test, override the setup() method

To release object(s) under test override the teardown() method

Determine one or more public test XYZ() methods that exercise the objects under test and assert expected results.

4)Some useful JUnit extensions?

JUnit extensions include

Cactus,

JWebUnit,

XMLUnit,

MockObject.

5) Annotations for Junit testing:

The Junit 4.x framework is annotation based, so let's see the annotations that can be used while writing the test cases.

@Test annotation specifies that method is the test method.

@Test(timeout=1000) annotation specifies that method will be failed if it takes longer than 1000 milliseconds (1 second).

@BeforeClass annotation specifies that method will be invoked only once, before starting all the tests.

@Before annotation specifies that method will be invoked before each test.

@After annotation specifies that method will be invoked after each test.

@AfterClass annotation specifies that method will be invoked only once, after finishing all the tests.

6)To run JUnit from the command window

To run JUnit from the command window, you have to follow the steps

Set the CLASSPATH

Invoke the runner:

Java org.junit.runner.JUnitCore

7)Test Suites: A test suite bundles a few unit test cases and runs them together. In JUnit, both @RunWith and @Suite annotation are used to run the suite test.

1.void addTest(Test test): Adds a test to the suite.

2.void addTestSuite(Class<? extends TestCase> testClass): Adds the tests from the given class to the suite.

3.int countTestCases(): Counts the number of test cases that will be run by this test.

4.String getName(): Returns the name of the suite.

5.void run(TestResult result): Runs the tests and collects their result in a TestResult.

6.void setName(String name): Sets the name of the suite.

7.Test testAt(int index): Returns the test at the given index.

8.int testCount(): Returns the number of tests in this suite.

9.static Test warning(String message): Returns a test which will fail and log a warning message.

Test Runners: Test runner is used for executing the test cases.

8) who should use JUnit – a developer or tester.

JUnit is more often used by developers to implement unit tests in JAVA. It is designed for unit testing that is more a coding process and not a testing process. However, many testers and QA engineers use JUnit for unit testing.

9) Why you use JUnit to test your code.

JUnit is used because

It test early and does automate testing

JUnit tests can be compiled with the build so that at unit level, regression testing can be done

It allows test code re-usage

JUnit tests behave as a document for the unit tests when there is a transfer

**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. Mockito facilitates creating mock objects seamlessly. It uses Java Reflection in order to create mock objects for a given interface. Mock objects are nothing but proxy for actual implementations.

Mockito adds a functionality to a mock object using the methods when().

Mockito can ensure whether a mock method is being called with required arguments or not. It is done using the verify() method.

Mockito provides Inorder class which takes care of the order of method calls that the mock is going to make in due course of its action.

Mockito provides the capability to a reset a mock so that it can be reused later.eg.//reset mock -> reset(calcService);

Behavior Driven Development is a style of writing tests uses given, when and then format as test methods. Mockito provides special methods to do so.

//Given

given(calcService.add(20.0,10.0)).willReturn(30.0);

//when

double result = calcService.add(20.0,10.0);

//then

Assert.assertEquals(result,30.0,0);

Mockito provides a special Timeout option to test if a method is called within stipulated time frame.//passes when add() is called within 100 ms.

verify(calcService,timeout(100)).add(20.0,10.0);

Mockito provides the following additional methods to vary the expected call counts.

atLeast (int min) − expects min calls.

atLeastOnce () − expects at least one call.

atMost (int max) − expects max calls.

Mockito provides the capability to a mock to throw exceptions, so exception handling can be tested.

Mockito provides a Answer interface which allows stubbing with generic interface.

Mockito provides option to create spy on real objects. When spy is called, then actual method of real object is called. eg.//create a spy on actual object - >calcService = spy(calculator);

Benefits of Mockito:

No Handwriting − No need to write mock objects on your own.

Refactoring Safe − Renaming interface method names or reordering parameters will not break the test code as Mocks are created at runtime.

Return value support − Supports return values.

Exception support − Supports exceptions.

Order check support − Supports check on order of method calls.

Annotation support − Supports creating mocks using annotation.