Important Interview Questions

**Question 1. What Is Mockito?**

**Answer:** Mockito allows creation of mock object for the purpose of Test Driven Development and Behavior Driven development. Unlike creating actual object, Mockito allows creation of fake object (external dependencies) which allows it to give consistent results to a given invocation.

**Question 2. Which Of The Following Are Usually Automated And Which Are Executed Manually?  
1. Unit Test  
2. Integration Test**

**Answer:** Unit Test are usually automated and Integration Tests are usually executed manually.

**Question 3. How To Create A Junit To Make Sure That The Tested Method Throws An Exception?**

**Answer:** Using annotation Test with the argument as expected exception. @Test (expected = Exception.class)

**Question 4. What Is The Use Of Mockito.any?**

**Answer:** In case we need to verify that a method is being called with any argument and not a specific argument we can use Mockito.any(Class), Mockito.anyString, Mockito.anyLong etc.

**Question 5. Why not just use System.out.println() for testing ?**

**Answer:** Debugging the code using system.out.println() will lead to manual scanning of the whole output every time the program is run to ensure the code is doing the expected operations. Moreover, in the long run, it takes lesser time to code JUnit methods and test them on class files.

**Question 6. Why does JUnit only report the first failure in a single test?**

**Answer:** 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.

**Question 7. What are the core features of JUnit?**

**Answer:** JUnit test framework provides following important features −

* Fixtures
* Test suites
* Test runners
* JUnit classes

**Question 8. What is a fixture?**

**Answer:** Fixture is a fixed state of a set of objects used as a baseline for running tests. The purpose of a test fixture is to ensure that there is a well known and fixed environment in which tests are run so that results are repeatable. It includes following methods −

* setUp() method which runs before every test invocation.
* tearDown() method which runs after every test method.

**Question 9. What are Parameterized tests?**

**Answer:** Junit 4 has introduced a new feature Parameterized tests. Parameterized tests allow developer to run the same test over and over again using different values.

**Question 10. What happens if a JUnit Test Method is Declared as "private"?**

**Answer:** If a JUnit test method is declared as "private", it compiles successfully. But the execution will fail. This is because JUnit requires that all test methods must be declared as "public".

**Question 11. What happens if a JUnit test method is declared to return "String"?**

**Answer:** If a JUnit test method is declared to return "String", the compilation will pass ok. But the execution will fail. This is because JUnit requires that all test methods must be declared to return "void".

**Question 12. Explain what is JUnitCore Class?**

**Answer:** JUnitCore class is an inbuilt class in JUnit package; it is based on Façade design pattern, this class is used to run only definite test classes only.

**Question 13. Explain how you can run JUnit from the command window?**

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

* Set the CLASSPATH
* Invoke the runner - Java org.junit.runner.JUnitCore

**Question 14. JUnit test files are written in files with which file extension?**

**Answer:** .java ----JUnit test files are regular java files with special methods which are referenced via annotations.

**Question 15. What does the fail() method do in JUnit?**

**Answer:** The method throws an assertion error unconditionally. This might be helpful to show an incomplete test (maybe still being worked upon) or to ensure that an expected exception is thrown.

**Question 16. Mention different methods of exception handling in JUnit?**

**Answer:** There are different methods of exception handling in JUnit

Try catch idiom

With JUnit rule

With @Test annotation

With catch exception library

With customs annotation

**Question 17. Explain what is ignore test in JUnit?**

**Answer:** When your code is not ready, and it would fail if executed then you can use @Ignore annotation.It will not execute a test method annotated with @Ignore. It will not execute any of the test methods of test class if it is annotated with @Ignore

**Question 18. Is the use of 'main' method possible for unit testing?**

**Answer:** Yes

**Question 19. Is it necessary to write the test class to test every class?**

**Answer:** No

**Question 20. What is the test suite?**

**Answer:** 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.

**Question 21. What does test runner?**

**Answer:** The test runner is used to execute the test cases.

**Question 22. What does Assert class?**

**Answer:** Assert class provides methods to test the test cases.

**Question 23. What Are The Limitations Of Mockito?**

**Answer:** Cannot mock static method. Cannot mock constructors

**Question 24. Name some JUnit code coverage tools?**

**Answer:** Cobertura, EclEmma

**Question 25. Which method of TestSuite class returns the test at the given index?**

**Answer:** Test testAtintindex

**Question 26. What is the purpose of org.junit.TestResult class?**

**Answer:** A TestResult collects the results of executing a test case. It is an instance of the Collecting Parameter pattern. The test framework distinguishes between failures and errors. A failure is anticipated and checked for with assertions. Errors are unanticipated problems like an ArrayIndexOutOfBoundsException.

**Question 27. Which method of TestResult class runs a TestCase?**

**Answer:** void run(TestCase test) method runs a TestCase.

**Question 28. What is Manual testing?**

**Answer:** Executing the test cases manually without any tool support is known as manual testing.

**Question 29. Do You Mock Classes & Interfaces?**

**Answer:** Yes, the api is the same for mocking classes or interfaces.

**Question 30. What is the purpose of org.junit.Assert class?**

**Answer:** This class provides a set of assertion methods useful for writing tests. Only failed assertions are recorded.

**Question 31. What are called as test smells in relation with unit testing?**

**Answer:** Multiple assertions within one unit test, long-running unit tests etc.

**Question 32. Why Do We Need Mockito? What Are The Advantages?**

**Answer:** Mockito differentiates itself from the other testing framework by removing the expectation beforehand. So, by doing this, it reduces the coupling. Most of the testing framework works on the "expect-run-verify". Mockito allows it to make it "run-verify" framework. Mockito also provides annotation which allows to reduce the boilerplate code.

**Question 33. What Values Do Mocks Return By Default?**

**Answer:** In order to be transparent and unobtrusive all Mockito mocks by default return 'nice' values. For example: zeros, falseys, empty collections or nulls. Refer to javadocs about stubbing to see exactly what values are returned by default.

**Question 34. What is mocking and stubbing?**

**Answer:** Mocking is a feature where an object mimics like a real object. Stubbing are codes that are responsible for taking place of another component.

**Question 35. What are the three components of J-Unit5?**

**Answer:** JUnit Platform, JUnit Jupiter and JUnit Vintage.

**Question 36. Can I Mock Static Methods?**

**Answer:** No. Mockito prefers object orientation and dependency injection over static, procedural code that is hard to understand & change. If you deal with scary legacy code you can use JMockit or Powermock to mock static methods.

**Question 37. Can I Mock Private Methods?**

**Answer:** No. From the standpoint of testing... private methods don't exist.

**Question 38. How do you test a "private" method?**

**Answer:** When a method is declared as "private", it can only be accessed within the same class. So there is no way to test a "private" method of a target class from any test class. Hence you need to perform unit testing manually. Or you have to change your method from "private" to "protected".

**Question 39. What is a Unit Test Case?**

**Answer:** Unit test case is nothing but a combination of input data and expected output, which is defined to test the proper functionality of a individual test unit. It’s is just to test the behavior of the unit for a particular input data.

**Question 40. How Can We Test Methods Individually Which Are Not Visible Or Declared Private ?**

**Answer:** We can either increase their visibility and mark them with annotation @VisibleForTesting or can use reflection to individually test those methods.

**Question 41. Can You Explain A Mockito Framework?**

**Answer:** In Mockito, you always check a particular class. The dependency in that class is injected using mock object. So, for example, if you have service class, then the Dao class are injected as a mockDao. This enables us to check only the method of that given service class and whether they are performing as expected or not.

**Question 42. What is Testing?**

**Answer:** Testing is the process of checking the functionality of the application whether it is working as per requirements.

**Question 43. What is Automated testing?**

**Answer:** Taking tool support and executing the test cases by using automation tool is known as automation testing.

**Question 44. What are the disadvantages of manual testing?**

**Answer:** Following are the disadvantages of manual testing −

Time consuming and tedious − Since test cases are executed by human resources so it is very slow and tedious.

Huge investment in human resources − As test cases need to be executed manually so more testers are required in manual testing.

Less reliable − Manual testing is less reliable as tests may not be performed with precision each time because of human errors.

Non-programmable − No programming can be done to write sophisticated tests which fetch hidden information.

**Question 45. What are the advantages of automated testing?**

**Answer:** Following are the advantages of automated testing −

* **Fast** − Automation runs test cases significantly faster than human resources.
* **Less investment in human resources** − Test cases are executed by using automation tool so less tester are required in automation testing.
* **More reliable** − Automation tests perform precisely same operation each time they are run.
* **Programmable** − Testers can program sophisticated tests to bring out hidden information.

**Question 46. What are important features of JUnit?**

**Answer:** Following are the important features of JUnit −

* It is an open source framework.
* Provides Annotation to identify the test methods.
* Provides Assertions for testing expected results.
* Provides Test runners for running tests.
* JUnit tests can be run automatically and they check their own results and provide immediate feedback.
* JUnit tests can be organized into test suites containing test cases and even other test suites.
* JUnit shows test progress in a bar that is green if test is going fine and it turns red when a test fails.

**Question 47. What are the best practices to write a Unit Test Case?**

**Answer:** 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 postcondition.

There must be at least two unit test cases for each requirement: one positive test and one negative test. If a requirement has sub-requirements, each sub-requirement must have at least two test cases as positive and negative.

**Question 48. When are Unit Tests written in Development Cycle?**

**Answer:** Tests are written before the code during development in order to help coders write the best code.

**Question 49. Name the tools with which JUnit can be easily integrated.**

**Answer:** JUnit Framework can be easily integrated with either of the followings −

* Eclipse
* Ant
* Maven

**Question 50. 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.

**Question 51. Explain the execution procedure of the JUint test API methods?**

**Answer:** Following is how the JUnit execution procedure works −

* First of all method annotated as @BeforeClass execute only once.
* Lastly, the method annotated as @AfterClass executes only once.
* Method annotated as @Before executes for each test case but before executing the test case.
* Method annotated as @After executes for each test case but after the execution of test case.
* In between method annotated as @Before and method annotated as @After each test case executes.

**Question 52. How to simulate timeout situation in JUnit?**

**Answer:** Junit provides a handy option of Timeout. If a test case takes more time than specified number of milliseconds then Junit will automatically mark it as failed. The timeout parameter is used along with @Test annotation.

**Question 53. How to create Parameterized tests?**

**Answer:** There are five steps, that you need to follow to create Parameterized tests−

* Annotate test class with @RunWith(Parameterized.class).
* Create a public static method annotated with @Parameters that returns a Collection of Objects (as Array) as test data set.
* Create a public constructor that takes in what is equivalent to one "row" of test data.
* Create an instance variable for each "column" of test data.
* Create your tests case(s) using the instance variables as the source of the test data.
* The test case will be invoked once per each row of data. Let's see Parameterized tests in action.

**Question 54. When are tests garbage collected?**

**Answer:** The test runner holds strong references to all Test instances for the duration of the test execution. This means that for a very long test run with many Test instances, none of the tests may be garbage collected until the end of the entire test run. Explicitly setting an object to null in the tearDown() method, for example, allows it to be garbage collected before the end of the entire test run.

**Question 55. Name some of the JUnit Extensions.**

**Answer:** Following are the JUnit extensions −

* Cactus
* JWebUnit
* XMLUnit
* MockObject

**Question 56. What is the purpose of @Test annotation in JUnit?**

**Answer:** The Test annotation tells JUnit that the public void method to which it is attached can be run as a test case.

**Question 57. What is the purpose of @Before annotation in JUnit?**

**Answer:** Several tests need similar objects created before they can run. Annotating a public void method with @Before causes that method to be run before each Test method.

**Question 58. What is the purpose of @After annotation in JUnit?**

**Answer:** If you allocate external resources in a Before method you need to release them after the test runs. Annotating a public void method with @After causes that method to be run after the Test method.

**Question 59. What is the purpose of @BeforeClass annotation in JUnit?**

**Answer:** Annotating a public static void method with @BeforeClass causes it to be run once before any of the test methods in the class.

**Question 60. What is the purpose of @AfterClass annotation in JUnit?**

**Answer:** This will perform the method after all tests have finished. This can be used to perform clean-up activities.

**Question 61. Name few Java Mocking frameworks ?**

**Answer:** Mockito, PowerMock, EasyMock, JMock, JMockit