* 1. **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.

* 1. **Question 2. 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.

* 1. **Question 3. 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.

For example, suppose, service class has an updateObject Method. That update method is depend on Dao1, Dao2. Here the objective is to test only the updateObject method and not the Dao1 and Dao2. So, we can create a mockDao and return a mock object. This object is not the one which is in the database but is custom made. It only tests whether the updates are happening as expected or not. The syntax to create a Mock object is as follows:

private static LedgerBook mockLedgerBook;  
In the setup(), you create the mock object  
setup(){  
mockLedgerBook = Mockito.mock(LedgeBookDao.class);  
}  
In the test method, you call this mockLedgerBook as follows:  
LedgerBook ledgerBook = new LedgerBook();  
ledgerBook.setName("default");  
Mockito.when(mockLedgerBook.findBookById(Mockito.anyLong()).thenReturn(ledgerBook);

* 1. **Question 4. What Is The Version You Used For Mockito Framework?**

**Answer :**

1.9.5 version of Mockito.

* 1. **Question 5. How Do You Create A Mockdb Object In Mockito?**

**Answer :**

We need to create the mock object and need to pass the invocation. We must override the method in the actual DB with the method over here and pass the response object.

 Mockito.when(mockObjectDao.find((Guid) Mockito.anyObject())).thenAnswer(new Answer<responseObject>() {  
            @Override  
            public responseObject answer(InvocationOnMock invocation) throws Throwable {  
                Guid responseObjectID = (Guid) invocation.getArguments()[0];  
                responseObject responseDbObject = new responseObject();  
                responseDbObject.setresponseObjectId(responseObjectID);  
                responseObjectLink responseObjectLink = new responseObjectLink();  
                responseObjectLink.setLedgerBookId(Guid.createGuid());  
                responseObjectLink.setresponseObjectId(responseObjectID);  
                responseObjectLink.setLedgerObjectId(Guid.createGuid());  
                responseDbObject.getresponseObjectLinks().add(responseObjectLink);  
                for (Contact sourceContact : responseObjectDTO.getSourceContacts()) {  
                    if (!sourceContact.getObjectId().equals(responseObjectID)) {  
                        responseObjectLink = new responseObjectLink();  
                        responseObjectLink.setLedgerBookId(sourceContact.getAddressBookId());  
                        responseObjectLink.setresponseObjectId(responseObjectID);  
                        responseObjectLink.setLedgerObjectId(sourceContact.getObjectId());  
                        responseDbObject.getresponseObjectLinks().add(responseObjectLink);  
                    }  
                }  
                return responseDbObject;  
            }  
        });

* 1. **Question 6. 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.

* 1. **Question 7. What Are Junits ?**

**Answer :**

Junt is the unit testing framework for Java.

* 1. **Question 8. Are Junits Tested Manually?**

**Answer :**

No , they are executed automatically.

* 1. **Question 9. How To Test Whether The Returns Value Of The Method Is Expected ?**

**Answer :**

Using Assert.

* 1. **Question 10. What Happens If The Assert Doesn't Evaluate To Be True ?**

**Answer :**

Junit fails.

* 1. **Question 11. 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)

* 1. **Question 12. What Should I Do If I Want To Make Sure That A Particular Method Of A Class Is Getting Called?**

**Answer :**

If its a static method of the class , we can use verify to make sure it's getting called.

If its an instance method , We can mock the object and then use verify with the mocked object to make sure that the method is getting called."

* 1. **Question 13. Name Few Java Mocking Frameworks?**

**Answer :**

Mockito, PowerMock, EasyMock, JMock, JMockit

* 1. **Question 14. 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.

* 1. **Question 15. How Should We Ignore Or Avoid Executing Set Of Tests ?**

**Answer :**

We can remove @Test from the respective test so as to avoid its execution. Alternatively we can put @Ignore annotation on the Junit file if we want to ignore all tests in a particular file.

* 1. **Question 16. 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.

* 1. **Question 17. Is It Really A Mocking Framework?**

**Answer :**

There is a bit of confusion around the vocabulary. Technically speaking, Mockito is a Test Spy framework. Usually developers use Mockito instead of a mocking framework. Test Spy framework allows to verify behaviour (like mocks) and stub methods (like good old hand-crafted stubs).

* 1. **Question 18. Why Is Mockito So Simple?**

**Answer :**

To promote simple test code that hopefully pushes the developer to write simple and clean application code. I wrote this paragraph long before version 1.5. Mockito is still quite lean but the number of features increased because many users found valid cases for them.

* 1. **Question 19. What Are The Limitations Of Mockito?**

**Answer :**

Mockito 2.x specific limitations

* + - Requires Java 6+
    - Cannot mock static methods
    - Cannot mock constructors
    - Cannot mock equals(), hashCode(). Firstly, you should not mock those methods. Secondly, Mockito defines and depends upon a specific implementation of these methods. Redefining them might break Mockito.
    - Mocking is only possible on VMs that are supported by Objenesis. Don't worry, most VMs should work just fine.
    - Spying on real methods where real implementation references outer Class via OuterClass.this is impossible. Don't worry, this is extremely rare case.

Mockito 1.x Specific limitations

* + - Needs Java 5+
    - Cannot mock final classes
    - Cannot mock final methods - their real behavior is executed without any exception. Mockito cannot warn you about mocking final methods so be vigilant.
    - Cannot mock static methods
    - Cannot mock constructors
    - Cannot mock equals(), hashCode(). Firstly, you should not mock those methods. Secondly, Mockito defines and depends upon a specific implementation of these methods. Redefining them might break Mockito.
    - Mocking is only possible on VMs that are supported by Objenesis (Note Objenesis is in version 2.1). Don't worry, most VMs should work just fine.
    - Spying on real methods where real implementation references outer Class via OuterClass.this is impossible. Don't worry, this is extremely rare case.
  1. **Question 20. Do You Mock Classes & Interfaces?**

**Answer :**

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

* 1. **Question 21. 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.

* 1. **Question 22. 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.

* 1. **Question 23. Can I Mock Private Methods?**

**Answer :**

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

* 1. **Question 24. Is Mockito Thread-safe?**

**Answer :**

For healthy scenarios Mockito plays nicely with threads. For instance, you can run tests in parallel to speed up the build. Also, you can let multiple threads call methods on a shared mock to test in concurrent conditions. Check out a timeout() feature for testing concurrency.

However Mockito is only thread-safe in healthy tests, that is tests without multiple threads stubbing/verifying a shared mock. Stubbing or verification of a shared mock from different threads is NOT the proper way of testing because it will always lead to intermittent behavior. In general, mutable state + assertions in multi-threaded environment lead to random results. If you do stub/verify a shared mock across threads you will face occasional exceptions like: WrongTypeOfReturnValue, etc.

* 1. **Question 25. Can I Verify Tostring()?**

**Answer :**

No. You can stub it, though. Verification of toString() is not implemented mainly because:

* + - When debugging, IDE calls toString() on objects to print local variables and their content, etc. After debugging, the verification of toString() will most likely fail.
    - toString() is used for logging or during string concatenation. Those invocations are usually irrelevant but they will change the outcome of verification.
  1. **Question 26. Can I "reset" A Mock?**

**Answer :**

Recently we decided to go on with this feature for tricky scenarios where mocks are created by the container. Before that, the lack of a reset method was deliberate to promote good testing habits and to make the API simpler.

Instead of reset() please consider writing simple, small and focused test methods over lengthy, over-specified tests.

* 1. **Question 27. What Are Unfinished Verification/stubbing Errors?**

**Answer :**

**Mockito validates if you use it correctly all the time. Examples of incorrect use:**

//Oups, someone forgot thenReturn() part:  
  when(mock.get());  
  //Oups, someone put the verified method call inside verify() where it should be outside:  
  verify(mock.execute());  
  //Oups, someone has used EasyMock for too long and forgot to specify the method to verify:  
  verify(mock);

Mockito throws exceptions if you misuse it so that you will know if your tests are written correctly. The only problem is that Mockito does the validation next time you use the framework. Therefore sometimes the exception is thrown in the next test and you have to manually find the previous test that was not written correctly.

* 1. **Question 28. Why Does Mockito Keep Threadlocal State?**

**Answer :**

Mockito uses ThreadLocal state to implement a gorgeous mocking syntax in a language full of constraints (yes, it's java). Fortunately, every time you interact with Mockito framework it validates the ThreadLocal state in case you misused the api.

* 1. **Question 29. Can I Thenreturn() An Inlined Mock() ?**

**Answer :**

Unfortunately you cannot do this:

  when(m.foo()).thenReturn(mock(Foo.class));  
  //                         ^

The reason is that detecting unfinished stubbing wouldn't work if we allow above construct. We consider this as a 'trade off' of framework validation (see also previous FAQ entry). However you can slightly change the code to make it working:

  //extract local variable and start smiling:

  Foo foo = mock(Foo.class);  
  when(m.foo()).thenReturn(foo);

* 1. **Question 30. Can I Stub Chained Getters?**

**Answer :**

sort of stubbing, e.g. mock to return mock, to return mock, etc. should be used very sporadically, ideally never. It clearly points out violation of the Law of Demeter. You don't want to mess with Demeter. Since you have been warned check out Mockito deep stubs.

* 1. **Question 31. Will This Library Help Mocking Closed (non-open) Classes?**

**Answer :**

Mockito-Kotlin does not really add any extra functionality to Mockito. As such, you will still get exceptions when trying to mock closed classes. If possible, try to use interfaces instead.

Mockito provides an incubating, opt-in feature to enable mocking closed classes.

* 1. **Question 32. Will This Library Help Overcoming Nullpointerexceptions With Mockito.any()?**

**Answer :**

Passing Mockito.any() to a non-nullable parameter in Kotlin fails with a NullPointerException. Mockito-Kotlin tries to overcome this.

* 1. **Question 33. I Get An Error ‘resolved Versions For App (1.1.x) And Test App (1.0.x) Differ’. What Should I Do?**

**Answer :**

Mockito-Kotlin depends on version 1.0.x of Kotlin’s standard and reflect library. If you depend on 1.1.x, you may get this error. To solve this, add the following snippet to your root’s build.gradle, replacing $kotlinVersion with the version your project uses:

subprojects {  
    configurations.all {  
        resolutionStrategy {  
            forcedModules = [  
              "org.jetbrains.kotlin:kotlin-stdlib:$kotlinVersion",  
              "org.jetbrains.kotlin:kotlin-reflect:$kotlinVersion"  
             ]  
        }  
    }  
}

If you don’t want to force dependencies like this, you can alternatively depend on the newer version of kotlin-reflect explicitly, which will make Mockito-Kotlin use that version as well:

testCompile 'com.nhaarman:mockito-kotlin:x.x.x`  
testCompile 'org.jetbrains.kotlin:kotlin-reflect:1.1.1'

* 1. **Question 34. Will This Library Support Mockito 1.x?**

**Answer :**

No. Mockito 2.x is officially released and this library will focus on these versions.

* 1. **Question 35. When Will This Library Reach 1.0.**

**Answer :**

Mockito-Kotlin reached 1.0.0 December 6th. Try it out!