Android Local Unit Test in Kotlin

Located At： module-name/src/**test**/java/

Run On : Local **JVM**

Dependencies : **JUnit**, **Mockito**, **Powermock**, Robolectric

Create Test : **Right-Click on TestSubject -> Go To -> Test** or **Alt + Enter -> Create Test**

Go TestSubject : **Right-Click on Test -> Go To -> TestSubject** or **Add JavaDoc 'Unit tests for the implementation of [TestSubject]' and Left-Click it**

Run Test: Gradle Tasks->verification->testDebugUnitTest... or ./gradlew test

Get Reports : module-name/build/reports/tests/index.html

CodeCoverage :

- Jacoco 0.8.3 (excludes)
- module-name\build\reports\jacoco\testDebugUnitTestCoverage\html\index.html
- Analyze -> show coverage Data --> module-name\build\jacoco\testDebugUnitTest.exec

---

JUnit4

1. JUnit Assert
    - Assert.assertEquals(**String** message, String **expected**, String **actual**)
    - recommend **import org.junit.Assert.***
    - assertEquals(String message, **double** expected, **double** actual, **double** delta)
    - assertEquals(String message, **float** expected, **float** actual, **float** delta)
    - **assertTrue**(String message, boolean condition) **assertNull**(String message, boolean actual)
    - **(1)** assertSame -> '==' **(2)** assertEquals -> equals()
    - assertEquals((Mutable)List,List) ERROR -> for ((index, day) in days.withIndex())
    - assertArrayEquals **'arrays first differed at element [0]'**
    - assertThat(String reason, **T** actual, Matcher<? super **T**> matcher)
        - is not equalTo

        - equalToIgnoringCase containsString startsWith endsWith
        - nullValue notNullValue
        - greaterThan lessThan greaterThanOrEqualTo lessThanOrEqualTo
        - closeTo allOf anyOf
        - hasKey hasValue hasItem
        - **extends BaseMatcher**
2. JUnit Annotation
    - JUnit4 Class : [Static]@BeforeClass -> @Before -> @Test -> @After -> [Static]@AfterClass
    - Test Method : @Before -> @Test -> @After
    - @Test(expected = IOException::class.java, timeout = 100)
    - @Ignore
    - @RunWith(TestRunne::class)
        - @RunWith(Parameterized::class) + @Parameterized.Parameters + public Iterable static fun()+ Public Constructor
        - @RunWith(Suite::class) + @Site.SuiteClasses({xxx::class, xxx::class})
        - @RunWith(PowerMockRunner::class)
    - @Rule [Static]@ClassRule
    - @FixMethodOrder(MethodSorters.DEFAULT)
        - MethodSorters.DEFAULT
        - MethodSorters.NAME_ASCENDING
        - MethodSorters.JVM
3. JUnit Rule
    - @Rule **[Static]**@ClassRule **implements TestRule**
    - Timeout DisableOnDebug
4. TestSuite
    - @RunWith(Suite::class) + @Site.SuiteClasses({xxx::class, xxx::class})
5. Error: "Method ... not mocked"

Mockito

1. **Mock**
    - Mockito.mock(T::class.java);
    - @Mock
        - @RunWith(MockitoJUnitRunner::class)
        - @Before MockitoAnnotations.initMocks(this);
        - @get:Rule var rule = MockitoJUnit.rule()
2. **Stubbing** to specify how a mock should **behave**
    - when(mockObject.someMethod(any())).thenReturn("A")

- when(mockIterator.next(anyString())).thenReturn("A").thenReturn("B")
- when(mockIterator.next(anyString())).thenReturn("A", "B", "C")
- when(mockIterator.next(argThat(isValid()))).thenReturn("A")
- when(mock.someMethod("some arg")).thenThrow(RuntimeException(), NullPointerException())
- when(mockIterator.next()).thenThrow(NullPointerException()).thenReturn("B")
- whenever(mockObject.someMethod(any(),eq("some arg"))).thenAnswer {

  doSomething() ...

  it.mock.toString() + "called " + it.method + " with arguments: " + Arrays.toString(it.arguments)
  }
- Stubbing **void** methods

  doReturn(Object) doThrow(Class or Throwable...) doAnswer(Answer) doNothing() doCallRealMethod()
  - stub void methods
  - stub methods on spy objects (see below)
  - stub the same method more than once, to change the behaviour of a mock in the middle of a test.
- var mock = mock(T::class.java.java, **TAnswer()**)
  - Mockito.RETURNS_DEFAULTS
  - Mockito.RETURNS_SMART_NULLS
  - Mockito.RETURNS_MOCKS
  - Mockito.RETURNS_DEEP_STUBS
  - Mockito.CALLS_REAL_METHODS
  - Mockito.RETURNS_SELF
- **ArgumentCaptor** vs **ArgumentMatcher.argThat(isValid())**
  - custom argument matcher is **not** likely to be **reused**
  - you **just** need it to assert on argument values to complete **verification**

3. **invocations**
   - verify(mockA).someMethod()
   - verify(mockA,verificationMode).someMethod()
     - timeout(100).times(3).description("someMethod should be called 3")
     - times(3) atMost(3) atLeast(2) never() atLeastOnce()
     - verifyZeroInteractions(mockB, mockC) verifyNoMoreInvocations(mockA) only()
   - inOrder.verify(mockA).someMethod()

     inOrder.verify(mockB).someMethod()
   - Argument Captors
4. Spying on real objects Real partial mocks
5. With **Kotlin**, Enable the option to mock final classes : **mock-maker-inline** or 'mark `Method` as `open`'
6. **TestMe IntelliJ Plugin**

Mockito-Kotlin

Mockito-Kotlin is a wrapper library around Mockito.

- It provides top-level functions to allow for a more idiomatic approach while using Mockito in Kotlin.
- Furthermore, Mockito returns null values for calls to method like any(), which can cause IllegalStateException when passing them to non-nullable parameters. This library solves that issue by trying to create actual instances to return.

1. **Creating mocks**
   - val mock : MyClass = mock()
   - val mock = mock()
   - myClass.test(mock())
2. **Stubing**
   - whenever(mock.stringValue()).thenReturn("test")
   - val mock = mock {
     on { stringValue() }.doReturn("test")
     }
   - val mock = mock {
     on { stringValue() } doReturn "test"
     }
3. **Verifying**
   - verify(myClass).doSomething("test")
   - verify(myClass).doSomething(any())
   - verify(myClass).doSomething(any())
   - For generic arrays, use `anyArray()`
4. **Properties**
   - verify(foo).bar = "test"
5. **Argument Matchers**
   - verify(myClass).setItems(argThat { size == 2 })
   - verify(myClass).setItems(argForWhich { size == 2 })
   - verify(myClass).setItems(check {
     assertThat(it.size, is(2))
     assertThat(it[0], is("test"))
     })
6. **Argument Captors**
   - argumentCaptor().apply {
     verify(myClass, times(2)).setItems(capture())

     assertEquals(2, allValues.size)
     assertEquals("test", firstValue)
     }
7. **InOrder**

- val a = ...
  val b = ...

  inOrder(a,b) {
  verify(a).doSomething()
  verify(b).doSomething()
  }

Powermock

1. Motivation
   - Using PowerMock, it becomes possible to mock static methods, remove static initializers, allow mocking without dependency injection, and more.
   - PowerMock does these tricks by modifying the byte-code at run-time when the tests are being executed
   - Scenarios : Using a 3rd party or legacy framework
2. Supported versions with Mockito 2
3. All usages require `@RunWith(PowerMockRunner::class)` and `@PrepareForTest` annotated at class level.
4. By default PowerMock loads all classes with its MockClassLoader.The classloader loads and modified all classes except:
   - system classes. They are deferred to system classloader
   - classes located in packages that are specified as ignored.
     - @PowerMockIgnore({ "org.mockito.", "*org.robolectric.SomeClass*", "*android.*" })
     - powermock.global-ignore= "org.myproject.", "*org.robolectric.*", "android.*"
5. **@PrepareForTest(T::class)** -- JavaDoc
   - Classes needed to be defined using this annotation are typically those that needs to be byte-code manipulated.
6. **Mocking Static Method**
   - @PrepareForTest(Static::class) // Static.class contains static methods
   - PowerMockito.mockStatic(Static::class.java)
   - whenever(Static.firstStaticMethod(param)).thenReturn(value)
7. **How to verify behavior**
   - PowerMockito.verifyStatic(Static::class.java) // 1

     Static.firstStaticMethod(param) // 2

     Static.thirdStaticMethod(Mockito.anyInt()) // 2
   - **Important: You need to call `verifyStatic(Static::class)` per static method verification.**
   - PowerMockito.verifyStatic(Static::class.java, Mockito.times(1))
   - PowerMockito.doThrow(ArrayStoreException("Mock error")).when(StaticService::class.java)

     StaticService.executeMethod()

- PowerMockito.doThrow(ArrayStoreException("Mock error")).when(myFinalMock).myFinalMethod()

8. **For private methods**
   - PowerMockito.when(tested, "methodToExpect", argument).thenReturn(myReturnValue)
   - PowerMockito.verifyPrivate(tested).invoke("privateMethodName", argument1)
9. **Mock construction of new objects （No Code Coverage）**
   - PowerMockito.whenNew(T::class.java).withAnyArguments().thenReturn(mockT)
   - PowerMockito.whenNew(T::class.java).withNoArguments().thenReturn(mockT)
   - PowerMockito.whenNew(T::class.java).withNoArguments().thenThrow(new IOException("error message"))
   - **Note that you must prepare the class *creating* the new instance of `MyClass` for test, not the `MyClass` itself.**
   - PowerMockito.verifyNew(T::class.java).withNoArguments()
10. **Bypass encapsulation** (Whitebox) -- Reflection
    - Use `Whitebox.setInternalState(..)` to set a non-public member of an instance or class.
    - Use `Whitebox.getInternalState(..)` to get a non-public member of an instance or class.
    - Use `Whitebox.invokeMethod(..)` to invoke a non-public method of an instance or class.
    - Use `Whitebox.invokeConstructor(..)` to create an instance of a class with a private constructor.
11. **Suppressing Unwanted Behavior**
    - Use the `@RunWith(PowerMockRunner::class)` annotation at the class-level of the test case.
    - Use the `@PrepareForTest(ClassWithEvilParentConstructor::class)` annotation at the class-level of the test case in combination with `suppress(constructor(EvilParent::class))` to suppress all constructors for the EvilParent class.
    - Use the `Whitebox.newInstance(ClassWithEvilConstructor::class)` method to instantiate a class without invoking the constructor what so ever.
    - Use the `@SuppressStaticInitializationFor("org.mycompany.ClassWithEvilStaticInitializer")` annotation to remove the static initializer for the the `org.mycompany.ClassWithEvilStaticInitializer` class.
    - Use the `@PrepareForTest(ClassWithEvilMethod::class)` annotation at the class-level of the test case in combination with `suppress(method(ClassWithEvilMethod::class, "methodName"))` to suppress the method with name "methodName" in the ClassWithEvilMethod class.
    - Use the `@PrepareForTest(ClassWithEvilField::class)` annotation at the class-level of the test case in combination with `suppress(field(ClassWithEvilField::class, "fieldName"))` to suppress the field with name "fieldName" in the ClassWithEvilField class.
12. **Test Listeners**
    - Use the `@RunWith(PowerMockRunner::class)` annotation at the class-level of the test case.
    - Use the `@PowerMockListener({AnnotationEnabler::class, FieldDefaulter::class, Listener2::class})` annotation at the class-level of the test case.
    - @PowerMockListener(AnnotationEnabler::class)
    - @PowerMockListener(FieldDefaulter::class)
13. **Mock Policies**
    - Use the `@RunWith(PowerMockRunner::class)` annotation at the class-level of the test case.

- Use the `@MockPolicy(MyMockPolicy::class)` annotation at the class-level of the test case.
- @MockPolicy(Slf4jMockPolicy.class)

14. **Mock System Classes** （No Code Coverage）
    - If you need to mock classes loaded by the java system/bootstrap classloader (those defined in the java.lang or java.net or java.io etc)
    - Use the `@RunWith(PowerMockRunner::class)` annotation at the class-level of the test case.
    - Use the `@PrepareForTest({ClassThatCallsTheSystemClass::class})` annotation at the class-level of the test case.
    - Use `mockStatic(SystemClass::class)` to mock the system class then setup the expectations as normally.
15. **Using @PowerMockRunnerDelegate**
    - @PowerMockRunnerDelegate(MockitoJUnitRunner::class)
    - @PowerMockRunnerDelegate(Parameterized::class)
16. @get:Rule var rule = PowerMockRule()
17. java.lang.Exception: No tests found matching Method xxx()
18. Mockito mock-maker-inline
19. Code coverage with JaCoCo
20. **FAQ**

Unit Testing with Retrofit + RxJava + MVVM

1. MockRetrofitHelper
    - OkHttp with PowerMockito java.lang.AssertionError: No System TLS -> @PowerMockIgnore("javax.net.ssl.*")
    - .registerTypeAdapter(Date::class.java, DateGsonAdapter(AppConfig.DEFAULT_DATE_TIME_FORMAT))
2. MVVM RxJava PowerMock
    - LiveData Room ...

      @get:Rule

      var instantExecutorRule = InstantTaskExecutorRule()
    - Rxjava

      companion object {
      @ClassRule
      @JvmField
      val schedulers = RxImmediateSchedulerRule()
      }
    - @PowerMockRunnerDelegate(MockitoJUnitRunner::class) //this line allows you to use the powerMock
    - LiveData changed use LiveDataTestUtil
    - SingleLiveEvent .call() setValue(null)

- Mockito.verify<Observer>(eventObserver, Mockito.times(1)).onChanged(null)