Unit Testing

Mockito & PowerMock

Why Unit Testing?

Helps to meet requirements

Gives confidence while Refactoring

Integrates with CI

Drives better Design

Improves Quality

Idea of UT

- Test methods of one class in isolation without dependencies
- Write test cases to reproduce and fix bugs
- Execute frequently, execute faster

Structure

```
@Test

public void testProcessPackets() throws Exception {

        // Arrange

        // Act

        // Assert

}
```

Junit Annotations

@Test

@Before

@BeforeClass

@AfterClass

@After

@RunWith(Class)

@Parameterized.parameter

Terminology

Stub - an object that provides predefined answers to method calls,

- a controllable replacement for an existing dependency
- Do not assert against stub

Mock - Mocks are objects that simulate the behavior of real object..

- Assert against mock object

Spy - allow us to call all the normal methods of the object while still tracking every interaction

Why Mock?

- Break dependencies
- Parallel working
- Runs Unit Tests faster
- Eliminates need to maintain stubs
- Easier to setup real objects
- Test interactions & methods with no return value

Mocking Libraries

Mockito

JMock

EasyMock

PowerMock

Why Mockito?

Easy to use

Clean syntax - BDD style (Given-When-Then)

Supports return values & Exceptions

Flexible parameter handling

Excellent documentation

Mockito

- Framework for interaction testing
- Provides default values
- Allows to define only the interactions that are relevant for the test.

Potential Methods

mock(MyService.class)

spy(new LinkedList())

when(processMessage())

thenReturn(new MessagePacket()) - Multiple return values possible

doReturn(new MessagePacket())

thenThrow(new IllegalStateException())

doThrow(new RuntimeException())

verify(mock).method(args)

Mockito Annotations

@Mock

@Spy

@InjectMocks

@Captor

Matchers

- Allow flexible verification
- Custom matchers are possible

anyString()

anyInt()

any(Class<T> clazz)

```
1.  Bar bar = mock(Bar.class);
2.    when(bar.isValidFoo(any(Foo.class))).thenReturn(true);
```

Argument Capture

- Captures Argument values for assertion

```
@Test
public void verifyTheValueAddedToList() {
    List mockList = Mockito.mock(List.class);
    ArgumentCaptor<String> arg = ArgumentCaptor.forClass(String.class);

    mockList.add("one");
    Mockito.verify(mockList).add(arg.capture());

    assertEquals("one", arg.getValue());
}
```

Verify

times()

never()

atLeastOnce()

atMost()

verifyZeroInteractions()

verifyNoMoreInteractions()

InOrder

Takes care of order of method calls

```
//create an inOrder verifier for a single mock
InOrder inOrder = inOrder(calcService);

//following will make sure that add is first called then subtract is called.
inOrder.verify(calcService).add(20.0,10.0);
inOrder.verify(calcService).subtract(20.0,10.0);
```

Asserts - Use Hamcrest

```
// Hamcrest for equals check
assertThat(actual, is(equalTo(expected)));

// Hamcrest for not equals check
assertThat(actual, is(not(equalTo(expected))));

// Collection Matcher
assertThat(myList, empty());

// List Contains elements
assertThat(myList, contains(“One”));

// Array Size
assertThat(myArray, arrayWithSize(4));
```

Hamcrest Matchers

- allOf - matches if all matchers match (short circuits)
- anyOf - matches if any matchers match (short circuits)
- not - matches if the wrapped matcher doesn't match and vice
- equalTo - test object equality using the equals method
- is - decorator for equalTo to improve readability
- hasToString - test Object.toString
- instanceOf, isCompatibleType - test type
- notNullValue, nullValue - test for null
- sameInstance - test object identity
- hasEntry, hasKey, hasValue - test a map contains an entry, key or value
- hasItem, hasItems - test a collection contains elements
- hasItemInArray - test an array contains an element
- closeTo - test floating point values are close to a given value
- greaterThan, greaterThanOrEqualTo, lessThan, lessThanOrEqualTo
- equalToIgnoringCase - test string equality ignoring case
- equalToIgnoringWhiteSpace - test string equality ignoring differences in runs of whitespace
- containsString, endsWith, startsWith - test string matching

BDD Mockito

BDD style of writing tests uses **//given //when //then** comments as fundamental parts of test methods. This is the recommended approach to write test cases.

- given
- willReturn
- willThrow
- willDoNothing
- should
- shouldHaveZeroInteractions
- shouldHaveNoInteractions

```
import static org.mockito.BDDMockito.*;

Seller seller = mock(Seller.class);
Shop shop = new Shop(seller);

public void shouldBuyBread() throws Exception {
  //given
  given(seller.askForBread()).willReturn(new Bread());

  //when
  Goods goods = shop.buyBread();

  //then
  assertThat(goods, containBread());
}
```

Limitations

- Final Classes
- Enums
- Private, Static, Final methods
- equals() && hashCode()

PowerMock

PowerMock

- Extension API to support Mockito
- Works with Java Reflection API and byte code manipulation
- Has ability to mock final, static or private methods.
- Integrates with Robolectric -- Really??

Prepare for PowerMock

- @RunWith(PowerMockRunner.class)
- @PrepareForTest(fullyQualifiedNames = "com.mypackage.test.util.*")
- @PowerMockIgnore(“org.mockito”)

- fullyQualifiedNames - Array of fully qualified names, tells Powermock to prepare all types within the package for mocking

Potential Methods

- mock()
- spy()
- when()

Remember to use Powermock version of Mocikto API

```
import static org.powermock.api.mockito.PowerMockito.mock;
import static org.powermock.api.mockito.PowerMockito.spy;
import static org.powermock.api.mockito.PowerMockito.when;
```

Mocking Constructor & Final Methods

- MyReaderThread mockReaderThread = mock(MyReaderThread.class)
- whenNew(MyReaderThread .class).withNoArguments().thenReturn(mockReaderThread);
- MyReaderThread readerThread= new MyReaderThread ();
- verifyNew(MyReaderThread.class).withNoArguments();

Mocking Final Methods

- MyReaderThread mockReaderThread = mock(MyReaderThread.class)
- when(mockReaderThread .processMyPacktets()).thenReturn(MyMessage.YAMS_RESULT_OK);
- int result = mockReaderThread .processMyPacktets();
- Mockito.verify(mockReaderThread).processMyPacktets();
- assertThat(MyMessage.RESULT_OK, is(result));

Mocking Private Methods

- Get a spy object // Arrange
  - WriterThread writerThread= new WriterThread();
  - WriterThread spyWriterThread= spy(writerThread);
  - when(spyWriterThread, "processMyPackets").thenReturn(RESULT_OK);

- int result = spyWriterThread.start(); // Act

- verifyPrivate(spyWriterThread).invoke("processMyPackets"); // Assert
- assertThat(RESULT_OK, is(result));

Mocking Static Methods

- Register the class that has static methods with Powermock
  - mockStatic(MyStaticSettingsUtil.class)

- Set mock return values
  - when(MyStaticSettingsUtil.hasTilting()) .thenReturn(true);
  - doThrow(new RuntimeException()).when(MyStaticSettingsUtil.class);
  - MyStaticSettingsUtil.commitMessage(any(Unit.class));

- Assert & Verify
  - assertTrue(MyStaticSettingsUtil.hasTilting());
  - verifyStatic(Mockito.times(1))
  - MyStaticSettingsUtil.commitMessage(any(Unit.class));

PowerMock and Mockito Compatibility

https://github.com/powermock/powermock/wiki/Mockito

Mockito	PowerMock
2.8.9+	2.x
2.8.0 - 2.8.9	1.7.x
2.7.5	1.7.0RC4
2.4.0	1.7.0RC2
2.0.0-beta - 2.0.42-beta	1.6.5-1.7.0RC

Powermock and Robolectric

- Add Dependencies
  - `testCompile "org.powermock:powermock-module-junit4:1.7.1"`
  - `testCompile "org.powermock:powermock-module-junit4-rule:1.7.1"`
  - `testCompile "org.powermock:powermock-api-mockito:1.7.1"`
  - `testCompile "org.powermock:powermock-classloading-xstream:1.7.1”`

Sample Test Class

```
@RunWith(RobolectricTestRunner.class)
@Config(constants = BuildConfig.class)
@PowerMockIgnore({ "org.mockito.*", "org.robolectric.*", "android.*" }) // Do not instrument mockito & robolectric libraries
@PrepareForTest(Static.class)
public class DeckardActivityTest {

    @Rule
    public PowerMockRule rule = new PowerMockRule();  // Needed to start powermock

    @Test
    public void testStaticMocking() {
        PowerMockito.mockStatic(Static.class);
        Mockito.when(Static.staticMethod()).thenReturn("hello mock");

        assertTrue(Static.staticMethod().equals("hello mock"));
    }
}
```

Sample Test Class

```
import org.junit.runner.RunWith;
import org.powermock.core.classloader.annotations.PowerMockIgnore;
import org.powermock.modules.junit4.PowerMockRunner;
import org.powermock.modules.junit4.PowerMockRunnerDelegate;
import org.robolectric.RobolectricTestRunner;
import org.robolectric.annotation.Config;
 /**
    * Base class extended by every Robolectric test in this project.
    */
   @RunWith(PowerMockRunner.class)
   @PowerMockRunnerDelegate(RobolectricTestRunner.class)
   @Config(constants = BuildConfig.class,  sdk = 21)
   @PowerMockIgnore({"org.mockito.*", "org.robolectric.*", "android.*"})
   public abstract class RobolectricTest {
   }
```

Limitations

- Cannot mock system classes (java.net.*, java.lang.*) because of JVM restrictions.
- PowerMock integration is broken in Robolectric 3.1 and 3.2, but fixed in 3.3.
- For Robolectric 3.3, we need Android Studio 3.0 because of tighter integration with the tool chain, where the build system processes and merges the resources.

Reference

- https://github.com/powermock/powermock/wiki
- https://github.com/robolectric/robolectric/wiki/Using-PowerMock
- http://www.baeldung.com/intro-to-powermock
- https://metlos.wordpress.com/2012/09/14/the-dark-powers-of-powermock/
- http://robolectric.org/getting-started/