Let’s Drink mockito

- A short introduction to mock framework

Agenda

- Mock? Why?
- Mockito ?
- Mockito - how to drink it? - framework basics
- Mockito and Spring
- Mockito – drinking examples
- Mockito with threads
- Mockito - pros and cons
- What else to use?
- Rules to remember

*`

Mock? Why?

- Mock - a simulated object that mimics the behavior of a real object in controlled ways.

Mock? **Why?**

- Better and faster testing and tests (also TTM)
- Integrate different systems
- Simpler small box -> Better design
- Faster to find bugs -> higher quality
- Why bother asking?

*`

Mockito?

- is a mocking framework that tastes really good. It lets you write beautiful tests with clean & simple API. Mockito doesn't give you hangover because the tests are very readable and they produce clean verification errors (from code.google.com/p/mockito)

*`

Mockito, how to drink it? framework basics

Stub - Mockito can mock concrete classes, not only interfaces

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

//mock creation
LinkedList mockedList = mock(LinkedList.class);


//using mock object
mockedList.add("one");
mockedList.clear();


//verification
verify(mockedList).add("one");
verify(mockedList).clear();
```

- Once created, mock will remember all interactions. Then you can selectively verify whatever interaction you are interested in.
- By default, for all methods that return value, mock returns null, an empty collection or appropriate primitive/primitive wrapper value (e.g: 0, false, ... for int/Integer, boolean/Boolean, ...).
- Stubbing can be overridden: for example common stubbing can go to fixture setup but the test methods can override it. Please note that overridding stubbing is a potential code smell that points out too much stubbing
- Once stubbed, the method will always return stubbed value regardless of how many times it is called.
- Last stubbing is more important - when you stubbed the same method with the same arguments many times.

Mockito, how to drink it? framework basics

Spy - Mockito support test spies not just mocks

- *List spy = spy(new LinkedList());*
-
- *//optionally, you can stub out methods:*
- *when(spy.size()).thenReturn(100);*
-
- *//using the spy calls real methods*
- *spy.add("one");*
- *spy.add("two");*
-
- *//prints "one" - the first element of a list*
- *System.out.println(spy.get(0));*
-
- *//size() method was stubbed - 100 is printed*
- *System.out.println(spy.size());*
-
- *//optionally, you can verify*
- *verify(spy).add("one");*
- *verify(spy).add("two");*

Important note on spying real objects!
1. Sometimes it's impossible to use when(Object) for stubbing spies. Example:

```
   List list = new LinkedList();
   List spy = spy(list);

   //Impossible: real method is called so spy.get(0) throws IndexOutOfBoundsException (the list is yet empty)
   when(spy.get(0)).thenReturn("foo");

   //You have to use doReturn() for stubbing
   doReturn("foo").when(spy).get(0);
```

2. Mockito doesn't mock final methods so the bottom line is: when you spy on real objects + you try to stub a final method = trouble.

Mockito, how to drink it? framework basics

- **Argument matchers**
  - Stubbing Build in (anyInt(), eq(), anyString()...)
  - *when(mockedList.get(anyInt())).thenReturn("element");*

  - Stubbing Hamcrest (argThat(org.hamcrest.Matcher))
  - *when(mockedList.contains(argThat(isValid()))).thenReturn("element");*

  - ArgumentCaptor to capture argument values for further assertion
  - *public class ArgumentCaptor<T> extends java.lang.Object*
  - *ArgumentCaptor<Person> argument = ArgumentCaptor.forClass(Person.class);*
  - *verify(mock).doSomething(argument.capture());*
  - *assertEquals("John", argument.getValue().getName());*
  - Also possible in verification
  - *verify(mockedList).get(anyInt());*

*`

Mockito, how to drink it? framework basics

Verifying - verify()

- *mockedList.add("once");*
- *mockedList.add("twice");*
- *mockedList.add("twice");*
-
- *//following two verifications work exactly the same - times(1) is used by default*
- *verify(mockedList).add("once");*
-
- *//exact number of invocations verification*
- *verify(mockedList, times(2)).add("twice");*
-
- *//verification using never(). never() is an alias to times(0)*
- *verify(mockedList, never()).add("never happened");*
-
- *//verification using atLeast()/atMost()*
- *verify(mockedList, atLeastOnce()).add("three times");*
- *verify(mockedList, atLeast(2)).add("five times");*
- *verify(mockedList, atMost(5)).add("three times");*

- *verifyZeroInteractions(mockedList2)...*

Mockito, how to drink it? framework basics

- **Maintain order – inOrder()**

```
List firstMock = mock(List.class);
List secondMock = mock(List.class);

 //using mocks
 firstMock.add("was called first");
 secondMock.add("was called second");

 //create inOrder object passing any mocks that need to be verified in order
 InOrder inOrder = inOrder(firstMock, secondMock);

 //following will make sure that firstMock was called before secondMock
 inOrder.verify(firstMock).add("was called first");
 inOrder.verify(secondMock).add("was called second");
```

*`

Mockito, how to drink it? framework basics

- **Return value thenReturn()**
- *when(mock.someMethod("some arg")).thenReturn("foo");*
- **Stubbing voids requires doReturn()**
- *doReturn("bar").when(mock).foo();*

*`

Mockito, how to drink it? framework basics

- **Exception thenThrow()**

- *when(mock.someMethod("some arg"))*
- *.thenThrow(new RuntimeException())*

- **Stubbing voids requires doThrow()**

- *doThrow(new RuntimeException()).when(mockedList).clear();*
- *//following throws RuntimeException:*
- *mockedList.clear();*

Mockito, how to drink it? framework basics

- **Return statefull mock answer thenAnswer()**

```
when(mock.someMethod(anyString())).thenAnswer(new Answer() {
    Object answer(InvocationOnMock invocation) {
        Object[] args = invocation.getArguments();
        Object mock = invocation.getMock();
        return "called with arguments: " + args;
    }
});
```

- **Stubbing voids do Answer()**

```
  doAnswer(new Answer() {
     public Object answer(InvocationOnMock invocation) {
         Object[] args = invocation.getArguments();
         Mock mock = invocation.getMock();
         return null;
     }})
 .when(mock).someMethod();
```

Mockito, how to drink it? framework basics

- **Other**
  - Calling real method
  - *when(mock.someMethod()).thenCallRealMethod();*
  - BDD aliases (given)
  - Serializable mocks
  - *mock(List.class, withSettings().serializable());*
  - Annotations
  - *@Mock, @Captor, @Spy, @InjectMocks*
  - Verification with timeout
  - *verify(mock, timeout(100)).someMethod();*
  - Reset (try not to use to often)
  - *reset(mock)*

*`

Mockito and Spring

Hardcoded approach

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = {
        "classpath:applicationConfig.xml"
})
public class DoSthTest {


@Test
public void shouldDoSth() {}
// given
   HttpClient mockHttpClient = Mockito.mock(HttpClient.class);
   serviceToTest.setHttpClient(mockHttpClient);
   ...
}
```

Mockito pros and cons

- Pros
  - Mockito is almost everywhere (python, java, c++, .Net)
  - A Good Humane Interface for Stubbing
  - Class (not just Interface) Mocks
  - Supports Test Spies, not just Mocks
  - Better Void Method Handling
  - Easy to write
  - Easy to learn
  - Validation

- Cons
  - Difficult to read (solution -> use given, when, then approach)
  - AbstractTestCases maintanance (solution -> avoid it)
  - Verify vs asserts can be badly used (try to use asserts)
  - Hard to learn ArgumentMatcher (solution -> just learn it ;))
  - Limitations of framework (rare cases but cant do anything about it)
    - Cannot mock final classes
    - Cannot mock static methods
    - 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 equals(), hashCode(). But that you should not mock
    - Spying on real methods where real implementation references outer Class via OuterClass.this is impossible. Don't worry, this is extremely rare case.

*`

What else to use

- PowerMock (private, final,static methods)
- Jmockit (constructors and static methods mocking)
- Hamcrest (library of matcher objects (also known as constraints or predicates) allowing 'match' rules to be defined declaratively)
- ?

*`

General rules to remember

- Mock it outside your code
- If you cannot test your code -> then probably you should change it ;) cause its badly written
- Test first
- Only one concrete class , mock the rest
- Only mock your nearest neighbour (Law of Demeter -> dont talk with strangers)
- Think ;) and then write

Thank you for your attention

*`