Mockito pitfalls

While testing with Mockito works, there are quite a lot of pitfalls. Mockito is technically very complex and fails silently in a huge amount of cases. In other cases you get messages containing very technical gibberish from which the real problem cannot really be found.. And don't even think of doing multithreaded tests... For this reason I dislike that framework a lot, and I prefer writing my code in such a way that it becomes testable by normal Java means.

To prevent us from having to search for common problems here is a list of pitfalls.

My test is successful, but my method-under-test is not even called??

This can happen when you are stubbing other methods in the *same* class that you are testing the method in. So for instance:

```
@Test
public void test() {
  Definition d = mock(Definition.class);
  when(d.getName()).thenReturn("hi");
  Assert.assertFalse(d.isNameInvalid());
}
```

The mock for Definition, by default, has all its methods stubbed to return default values. Consequently the call to d.isNameValid() does **not** call the real method but the stub, and that returns false by default. Consequently the test works, but does not test anything.

Since you cannot know when this happens Mockito tests should be carefully made to make sure you do not call nonsense methods.

The solution to this is what is called a *partial mock* where you define which methods should be called despite the object being a mock:

```
when(d.isNameInvalid()).thenCallRealMethod();
```

The fun part is that this needs to be done for ALL methods in the mock that are called from d.isNameInvalid() too, otherwise these fake their results too.

Stubbing a method throws NPE from that method

Real funny too.. Take the following:

```
class Definition {
  private Thingy m_main;


  final public getName() {
    return m_main.getRealName();
  }
}


class Test {
  @Test
  public void test() {
    Definition d = mock(Definition.class);
    when(d.getName()).thenReturn("hello");
    ....
}
```

When run this throws a NPE from the Definition.getName() class as it is called through the when method.

The root cause is the **final** on the method (or the class), as this prevents Mockito from overriding the method properly, so the original method gets called - and that dies because m_main is null.

For some nonsense reason Mockito does not give a warning for this because the when() only does its work AFTER the method is called on the stub. It apparently never occurred to anyone to report an error while the mock is being made which would at least give an indication of WHY THE BLOODY THING FAILS.

To at least get an indication of wrongness do not use when(...).thenReturn but try the equivalent doReturn(...).when(...) like this:

```
  @Test
  public void test() {
    Definition d = mock(Definition.class);
    doReturn("hello").when(d).getName();
    ....
}
```

This at least gives an error message (example):

```
org.mockito.exceptions.misusing.UnfinishedStubbingException:
Unfinished stubbing detected here:
-> at nl.skarp.portal.test.sqlgen.OrderingRuleTest.orderingChildrenCanOnlyBePIorOO(OrderingRuleTest.java:67)

E.g. thenReturn() may be missing.
Examples of correct stubbing:
    when(mock.isOk()).thenReturn(true);
    when(mock.isOk()).thenThrow(exception);
    doThrow(exception).when(mock).someVoidMethod();
Hints:
 1. missing thenReturn()
 2. you are trying to stub a final method, which is not supported
 3: you are stubbing the behaviour of another mock inside before 'thenReturn' instruction if completed
```

In this case #2 tells the real issue: a final method being stubbed.

The usual solution is to remove the final, so code quality is less because we use a very technical tool.

Mockito 2 allows stubbing finals as described here.

See this stackoverflow answer for more details.