[Java samples & frameworks](http://nixx78.blogspot.com/)

Ads by OnlineBrowserAdvertising[Ad Options](http://asrv-a.akamaihd.net/sd/apps/adinfo-1.0-p/index.html?bj1PbmxpbmVCcm93c2VyQWR2ZXJ0aXNpbmcmaD1hc3J2LWEuYWthbWFpaGQubmV0JmM9Z3JlZW4mbz13c2FyJmQ9JnQ9MTsyOzM7NDs1OzY7Nzs4Ozk7MTA7MTE7MTI7MTM7MTQmYT0xNzAwJnM9MTAzNyZ3PW5peHg3OC5ibG9nc3BvdC5jb20mYj1iZDImcmQ9JnJpPQ==)

<http://nixx78.blogspot.com/p/blog-page.html>

Использование Mockito в Unit тестах

**Mockito framework: создания заглушек для Unit тестов**

**Назначение фреймворка**

Фреймворк Mockito предназначен для создание заглушек, которые можно использовать при написании Unit тестов вместо реальных классов или интерфейсов.   
Страница фреймворка: <http://code.google.com/p/mockito/>  
  
Наиболее интересными мне показались следующие возможности фреймворка:

 Создание заглушек как для классов так и для интерфейсов

 Возможность проверки, что метод вызывался и с какими параметрами

 Концепция "частичной заглушки", в этом случае заглушка создается на класс и есть возможность задать поведение требуемое для тестов только для некоторых методов

 Возможность повесить spy() на класс с реализацией, потом можно проверять с какими параметрами вызывались методы класса

**Библиотеки требуемые для работы примеров**

 Log4j - логгирование

 Mockito - фреймворк для создания/работы с заглушками

 Hamcrest - фреймворк с различными Matchers для упрощения тестирования

 JUnit - фреймворк для написания Unit тестов  
Проект с примерами находится тут:[https://github.com/nixx78/mockito-usage-samples](https://github.com/nixx78/mockito-usage-samples/)

**Примеры использования Mockito**

Все примеры находятся в классе MockitoUsageSamples, в том же пакете находятся классы для которых создаются заглушки.

**Самый простой пример**

Самый простой пример использования Mockito, мы создаем mock объект для интерефейса, вызываем у него методы, потом проверяем, действительно ли данные методы вызывались. При этом, мы можем проверить, параметры с которыми вызывались методы.

[?](http://nixx78.blogspot.com/p/blog-page.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | @Test  public void methodCallCheckSample() {   // create mock for interface   InterfaceForTest mockedObject = mock(InterfaceForTest.class);     // call methods   mockedObject.setStringValue("value1");   mockedObject.actionMethod();     // verify, that methods are called, please note that we check   // also parameter value   verify(mockedObject).setStringValue("value1");   verify(mockedObject).actionMethod();     // in this point we expect exception, because method "getStringValue" not called   //verify(mockedObject).getStringValue();  } |

**Чуть более сложный пример, добавим заглушки методов**

Пример, в котором мы рассматриваем как мы можем сделать стабы для методов, в стабах мы указываем, какое значение мы ожидаем от метода при его вызове с определенными параметрами. Показаны различные подходы, как можно задавать условия,при создании заглушек для методов.

[?](http://nixx78.blogspot.com/p/blog-page.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | @Test  public void stubUsageSample() throws Exception{     // create mocked object   InterfaceForTest mockedObject = mock(InterfaceForTest.class);     // describe expected behavior for our interface   when(mockedObject.getStringValue()).thenReturn("expectedValue");     // we can define for one method different return values depending from input value   when(mockedObject.processMethod("input1")).thenReturn("value1");   when(mockedObject.processMethod("input2")).thenReturn("value2");   when(mockedObject.processMethod("input3")).thenThrow(new Exception("exception during method 'processMethod' call"));     // also we can define expected value using another approach: doReturn   doReturn("value4").when(mockedObject).processMethod("input4");   doThrow(new Exception()).when(mockedObject).processMethod("method with exception parameter");     // assert expected values, that method returns   assertEquals("value1", mockedObject.processMethod("input1"));   assertEquals("value2", mockedObject.processMethod("input2"));     try {    mockedObject.processMethod("input3");   } catch (Exception ex) {    // we expect exception there    System.err.println(ex);   }     // for non mocked input value we expect null   assertNull(mockedObject.processMethod("not mocked value"));     assertEquals("getString value", "expectedValue", mockedObject.getStringValue());  } |

**Сложная заглушка для метода - используем интерфейс Answer**

Пример показывает пример реализации стабов для методов используя интерфейс Answer, данный подход позволяет реализовать заглушки методов со сложным поведением. Примером такого поведения может быть ситуация, когда ответ возвращаемый методом зависит не только от передаваемого параметра но и требует более сложной логики при его создании.

[?](http://nixx78.blogspot.com/p/blog-page.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | @Test   public void smartStubUsageSample() throws Exception {    // create mock object    InterfaceForTest mockedObject = mock(InterfaceForTest.class);      // create mocked for method using call back    when(mockedObject.processMethod(anyString())).thenAnswer(new Answer<string>() {       public String answer(InvocationOnMock invocation) {       // we can take method arguments as array of objects             Object[] args = invocation.getArguments();             return "called with arguments: " + args[0];         }      });      // calling method 'processMethod' we expect, that method 'answer' is called in mockedObject    String returnValue = mockedObject.processMethod(String.valueOf(System.currentTimeMillis()));    System.out.println(returnValue);   }  </string> |

**Особый случай при сравнение, сложный объект - создаем свой Matcher**

Пример реализации собственного матчера, это позволяет написать матчер для особых случаев и особых объектов.

[?](http://nixx78.blogspot.com/p/blog-page.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | @Test   public void customArgumentMatchersSample() throws Exception {      // create mock object    InterfaceForTest mockedObject = mock(InterfaceForTest.class);      // create custom matcher for method argument    Matcher<string> matcher = new ArgumentMatcher<string>() {     @Override     public boolean matches(Object argument) {      return argument.equals("expectedValue") || argument.equals("EXPECTED\_VALUE") ;     }    };      // assign matcher to the method call    when(mockedObject.processMethod(argThat(matcher))).thenReturn("returnedValue");      // assert expected values    assertEquals("returnedValue", mockedObject.processMethod("expectedValue"));    assertEquals("returnedValue", mockedObject.processMethod("EXPECTED\_VALUE"));    assertNull(mockedObject.processMethod("not mocked value"));     }  </string></string> |

**Стандартных матчеров недостаточно - используем Hamcrest**

Пример использования Hamcreast matcher, больше информации об этом фреймворке можно посмотреть на странице: http://code.google.com/p/hamcrest/wiki/Tutorial

[?](http://nixx78.blogspot.com/p/blog-page.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46 | @Test  public void hamrestMatcherUsageSample() throws Exception {     // create mock object   InterfaceForTest mockedObject = mock(InterfaceForTest.class);     // create stub for expected method calls   when(mockedObject.getStringValue()).thenReturn("\_value1");     // using Hamcrest mathcers check expected method return values   assertThat(mockedObject.getStringValue(), org.hamcrest.Matchers.endsWith("1") );   assertThat(mockedObject.getStringValue(), org.hamcrest.Matchers.startsWith("\_") );     /\*    \* Extract from Hamcrest JavaDoc:            Core            anything - always matches, useful if you don't care what the object under test is            describedAs - decorator to adding custom failure description            is - decorator to improve readability - see "Sugar", below        Logical            allOf - matches if all matchers match, short circuits (like Java &&)            anyOf - matches if any matchers match, short circuits (like Java ||)            not - matches if the wrapped matcher doesn't match and vice versa        Object            equalTo - test object equality using Object.equals            hasToString - test Object.toString            instanceOf, isCompatibleType - test type            notNullValue, nullValue - test for null            sameInstance - test object identity        Beans            hasProperty - test JavaBeans properties        Collections            array - test an array's elements against an array of matchers            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        Number            closeTo - test floating point values are close to a given value            greaterThan, greaterThanOrEqualTo, lessThan, lessThanOrEqualTo - test ordering        Text            equalToIgnoringCase - test string equality ignoring case            equalToIgnoringWhiteSpace - test string equality ignoring differences in runs of whitespace            containsString, endsWith, startsWith - test string matching    \*/  } |

**Проверим сколько раз вызывался метод, а также, какой был порядок вызовов методов**

Пример показывает как можно проверить количество вызовов методов а также порядок, в котором они вызываются.

[?](http://nixx78.blogspot.com/p/blog-page.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | @Test  public void methodInvocationCountAndOrder() {   // create mock object   InterfaceForTest mockedObject = mock(InterfaceForTest.class);     // call methods in mock   mockedObject.actionMethod();   mockedObject.actionMethod();     mockedObject.setStringValue("1");   mockedObject.setStringValue("2");     mockedObject.setStringValue("3");   mockedObject.setStringValue("3");   mockedObject.setStringValue("3");     mockedObject.setStringValue("4");     mockedObject.getStringValue();   mockedObject.getStringValue();     // verify method calls, also we check method invocation count   verify(mockedObject, times(2)).actionMethod();     verify(mockedObject, times(3)).setStringValue("3");   verify(mockedObject, atLeast(1)).getStringValue();     // create wrapper object, using this wrapper we can verify method's call order   InOrder inOrder = inOrder(mockedObject);   inOrder.verify(mockedObject).setStringValue("1");   inOrder.verify(mockedObject).setStringValue("2");   inOrder.verify(mockedObject, times(3)).setStringValue("3");   inOrder.verify(mockedObject).setStringValue("4");  } |

**Иногда правильный результат - ниодин метод не вызывался, проверим это**

Проверка, что ни один метод не вызывался.

[?](http://nixx78.blogspot.com/p/blog-page.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @Test  public void verifyZeroInteractionsCheck(){   InterfaceForTest mockedObject = mock(InterfaceForTest.class);     verifyZeroInteractions(mockedObject);  } |

**Проверка - вызываются только нужные методы**

Пример показывает, как проверить, что вызываются только необходимые методы, другие не вызываются.

[?](http://nixx78.blogspot.com/p/blog-page.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | @Test  public void verifyNoMoreInteractionsCheck() {     InterfaceForTest mockedObject = mock(InterfaceForTest.class);   mockedObject.actionMethod();     // verify, that method 'actionMethod' called   verify(mockedObject).actionMethod();     // verify, that method 'setStringValue()' never called   verify(mockedObject, never()).setStringValue("value");     // verify, that no more methods is called   verifyNoMoreInteractions(mockedObject);  } |

**Регестрируем spy() для класса, для некоторых методов определяем заглушку**

Пример использования для тестов реального класса. При этом, мы можем проверить, вызывался ли метод, сколько раз и с какими параметрами. Также, есть возможность создавать заглушки только для некоторых методов, это очень удобно, когда мы хотим проверить поведение одного метода, используя заглушку для других.

[?](http://nixx78.blogspot.com/p/blog-page.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | @Test  public void spyOnRealMethodAndPartialMocking() {   // create spy around real class, please note: we create real class instance   InterfaceForTest impl = spy(new InterfaceForTestImpl());     // set expected value only for one method   when(impl.getIntValue()).thenReturn(200);     // call method, method will be call on real class   impl.setStringValue("stringValue");     // we can verify, that method are called with expected input parameter   verify(impl).setStringValue("stringValue");     // in this case, we expect value set for mock, not real method call   assertEquals(200, impl.getIntValue() );  } |