Глоток МоКито

[JAVA](http://habrahabr.ru/hub/java/)\*

А Вы знаете что такое mock-объект? [Нет](http://azeffin.habrahabr.ru/blog/48712/)?  
  
Что же говорит [по этому поводу википедия](http://en.wikipedia.org/wiki/Mock_object): «В объектно-ориентированном программировании mock-объект имитирует поведение реального объекта заданным образом… ». Казалось бы зачем? Википедия [продолжает](http://en.wikipedia.org/wiki/Mock_object#Reasons_for_use): «Во время unit-тестирования mock-объекты могут симулировать поведение бизнес-объектов и бизнес-логику, что иногда необходимо из-за сложности реального поведения»  
  
  
И что же дают mock-библиотеки java-разработчику? Конечно же удобство создания и использования тех самых mock-объектов!   
  
java-source.net [приводит список](http://java-source.net/open-source/testing-tools) аж из 7 библиотек:

* [EasyMock](http://www.easymock.org/)
* [Mocquer](https://mocquer.dev.java.net/)
* [MockLib](http://mocklib.sourceforge.net/)
* [Mockrunner](http://mockrunner.sourceforge.net/index.html)
* [jMock](http://www.jmock.org/)
* [MockEJB](http://mockejb.sourceforge.net/)
* [MockCreator](http://mockcreator.sourceforge.net/)

Я начал знакомство с mock-объектами c другой библиотеки, которой нет на java-source.net — [Mockito](http://code.google.com/p/mockito/). Я просто доверился мнению [javaposse](http://www.javaposse.com/index.php?post_id=530959), признавших mockito библиотекой недели, а так же [настроению java-блогосферы](http://code.google.com/p/mockito/wiki/Quotes) и не разочаровался. Вы тоже оцените насколько это удобно:  
  
*Внимание! Сделав глоток — остановиться невозможно.*

Проверяем поведение

mock-объект будет запоминать любые вызовы его методов, чтобы после Вы могли проверить какие методы вызывал ваш тестируемый код у mock-объекта

[Copy Source](http://s-c.me/3462/s) | [Copy HTML](http://s-c.me/3462/h)

1. //статически импортируем методы (для красоты и легкости кода)
2. import static org.mockito.Mockito.\*;
4. //вот он - mock-объект (заметьте: List.class - это интерфейс)
5. List mockedList = mock(List.class);
7. //используем его
8. mockedList.add("one");
9. mockedList.clear();
11. //проверяем, были ли вызваны методы add с параметром "one" и clear
12. verify(mockedList).add("one");
13. verify(mockedList).clear();

А как же stub'ы?

По умолчанию все методы mock объекта возвращают default'ные значение, false для boolean, 0 для int, пустые коллекции, null для остальных объектов.

[Copy Source](http://s-c.me/3463/s) | [Copy HTML](http://s-c.me/3463/h)

1. //Вы можете создавать mock для конкретного класса, не только для интерфейса
2. LinkedList mockedList = mock(LinkedList.class);
4. //stub'инг
5. when(mockedList.get(0)).thenReturn("first");
6. when(mockedList.get(1)).thenThrow(new RuntimeException());
8. //получим "first"
9. System.out.println(mockedList.get(0));
11. //получим RuntimeException
12. System.out.println(mockedList.get(1));
14. //получим "null" ибо get(999) не был определен
15. System.out.println(mockedList.get(999));

Проверяем точное количество вызовов

[Copy Source](http://s-c.me/3464/s) | [Copy HTML](http://s-c.me/3464/h)

1. //используем mock-объект
2. mockedList.add(«once»);
4. mockedList.add(«twice»);
5. mockedList.add(«twice»);
7. mockedList.add(«three times»);
8. mockedList.add(«three times»);
9. mockedList.add(«three times»);
11. //по умолчанию проверка, что вызывался 1 раз ~ times(1)
12. verify(mockedList).add(«once»);
13. verify(mockedList, times(1)).add(«once»);
15. //точное число вызовов
16. verify(mockedList, times(2)).add(«twice»);
17. verify(mockedList, times(3)).add(«three times»);
19. //никогда ~ never() ~ times(0)
20. verify(mockedList, never()).add(«never happened»);
22. //как минимум, как максимум
23. verify(mockedList, atLeastOnce()).add(«three times»);
24. verify(mockedList, atLeast(2)).add(«five times»);
25. verify(mockedList, atMost(5)).add(«three times»);

Паразитируем на реальных объектах

Вы можете создать mock-объект, (точнее spy-объект) который будет использовать реальный объект при вызовах метода. Перегрузив лишь необходимые вам методы, так сказать частичный mock-объект. 

[Copy Source](http://s-c.me/3465/s) | [Copy HTML](http://s-c.me/3465/h)

1. List list = new LinkedList();
2. List spy = spy(list);
4. //опционально, определяем лишь метод size()
5. when(spy.size()).thenReturn(100);
7. //используем реальные методы
8. spy.add("one");
9. spy.add("two");
11. //получим "one"
12. System.out.println(spy.get(0));
14. //метод size() нами переопределён - получим 100
15. System.out.println(spy.size());
17. //можем проверить
18. verify(spy).add("one");
19. verify(spy).add("two");

А так же

* Используем маски в аргументах переопределяемых методов
* Переопределяем методы, возвращающие void
* Проверяем порядок вызовов mock-объекта
* Ищем ненужные вызовы mock-объекта
* Аннотация @Mock
* Различные возвращаемые значения, для одних и тех же аргументов
* … [остатки сладки](http://mockito.googlecode.com/svn/branches/1.8.0/javadoc/org/mockito/Mockito.html)

Дополнительная литература:

* [Сравнение Mockito и самой похожей на неё EasyMock](http://code.google.com/p/mockito/wiki/MockitoVSEasyMock)
* [Обзор библиотеки на InfoQ](http://www.infoq.com/news/2008/09/mockito-1.5)
* [Выбор лучшей mock-библиотеки на stackoverflow](http://stackoverflow.com/questions/22697/whats-the-best-mock-framework-for-java)
* [mockobjects.com](http://www.mockobjects.com/)