Mockito

- 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.

www.intelliware.ca — © 2009 Intelliware Development Inc.

Syntax

- **Let's keep it simple: interactions are method calls.** There is no point in building sophisticated mocking language for method calls in Java. There is already a language for that. It's called Java.
- **The less DSL the better.** Interactions are just method calls. Method calls are simple, DSL is complex.
- **No Strings for methods.** I spent more time reading code than writing it. Therefore I want the code to be readable. String literals pretending to be methods cannot compete with actual method calls. Even IDEs decorate literals differently so my mind will always adopt the real method call quicker than any literal trying to impersonate a method.

Syntax

- **No anonymous inner classes.** They bring more indentation, more curly brackets, more code, more noise. Did I mention that I spent more time reading code?
- **Painless refactoring.** Renaming a method should not break my tests.

intelliware.ca
software development

drinking Mockito gives you ...

- **Focused testing.** Should let me focus test methods on specific behavior/interaction. Should minimize distractions like expecting/recording irrelevant interactions.
- **Separation of stubbing and verification**. Should let me code in line with intuition: stub before execution, selectively verify interactions afterwards. I don't want any verification-related code before execution.
- **Explicit language.** Verification and stubbing code should be easy to discern, should distinguish from ordinary method calls / from any supporting code / assertions.
- **Simple stubbing model.** Should bring the simplicity of good old hand crafted stubs without the burden of actually implementing a class. Rather than verifying stubs I want to focus on testing if stubbed value is used correctly.

drinking Mockito gives you ...

- **Only one type of mock, one way of creating mocks.** So that it's easy to share setup.
- **No framework-supporting code.** No record(), replay() methods, no alien control/context objects. These don't bring any value to the game.
- **Slim API.** So that anyone can use it efficiently and produce clean code. API should push back if someone is trying to do something too smart: if the class is difficult to test – refactor the class.
- **Explicit errors.** Verification errors should always point stack trace to failed interaction. I want to click on first stack element and navigate to failed interaction.
- **Clean stack trace.** Part of TDDing is reading stack traces. It's Red-Green-Refactor after all – I've got stack trace when it's red. Stack trace should always be clean and hide irrelevant internals of a library. Although modern IDE can help here, I'd rather not depend on IDE neither on competence in using IDE…

Mockito's documentation

- **Clean and concise**
- **Cookbook style (has sample code)**
- **Up to date**
- **Covers all features**

Let's go over Mockito's documentation

Test doubles

When you want to test code that depends on something that is too difficult or slow to use in a test environment, swap in a test double for the dependency.

- **Dummy.** A Dummy passes bogus input values around to satisfy an API.
- **Stub.** A Stub overrides the real object and returns hard-coded values. Testing with stubs only is state-based testing; you exercise the system and then assert that the system is in an expected state.
- **Mock.** A Mock can return values, but it also cares about the way its methods are called ("strict mocks" care about the order of method calls, whereas "lenient mocks" do not.) Testing with mocks is interaction-based testing; you set expectations on the mock, and the mock verifies the expectations as it is exercised.
- **Spy.** A Spy serves the same purpose as a mock: returning values and recording calls to its methods. However, tests with spies are state-based rather than interaction-based, so the tests look more like stub style tests.
- **Fake.** A Fake swaps out a real implementation with a simpler, fake implementation. The classic example is implementing an in-memory database

Dummy

A Dummy passes bogus input values around to satisfy an API.

```
Item item = new Item(ITEM_NAME);
ShoppingCart cart = new ShoppingCart();
cart.add(item, QUANTITY);
assertEquals(QUANTITY, cart.getItem(ITEM_NAME));
```

intelliware.ca
software development

Stub

A Stub overrides the real object and returns hard-coded values. Testing with stubs only is state-based testing; you exercise the system and then assert that the system is in an expected state.

```
ItemPricer pricer = new ItemPricer() {
  public BigDecimal getPrice(String name){ return PRICE; }
};
ShoppingCart cart = new ShoppingCart(pricer);
cart.add(dummyItem, QUANTITY);
assertEquals(QUANTITY*PRICE, cart.getCost(ITEM_NAME));
```

intelliware.ca
software development

Mock

A Mock can return values, but it also cares about the way its methods are called (“strict mocks” care about the order of method calls, whereas “lenient mocks” do not.) Testing with mocks is interaction-based testing; you set expectations on the mock, and the mock verifies the expectations as it is exercised.

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

CheckoutManager manager = mock(CheckoutManager.class);
ShoppingCart cart = new ShoppingCart(manager);
cart.add(dummyItem, QUANTITY);
cart.checkout();

verify(manager).checkout(cart);
```

intelliware.ca
software development

Spy

A Spy serves the same purpose as a mock: returning values and recording calls to its methods. However, tests with spies are state-based rather than interaction-based, so the tests look more like stub style tests.

```
TransactionLog log = new TransactionLogSpy();
ShoppingCart cart = new ShoppingCart(log);
cart.add(dummyItem, QUANTITY);
assertEquals(1, logSpy.getNumberOfTransactionsLogged());
assertEquals(QUANTITY*PRICE, log.getTransactionSubTotal(1));
```

intelliware.ca
software development

Fake

A Fake swaps out a real implementation with a simpler, fake implementation. The classic example is implementing an in-memory database.

```
Repository repo = new InMemoryRepository();
ShoppingCart cart = new ShoppingCart(repo);
cart.add(dummyItem, QTY);
assertEquals(1, repo.getTransactions(cart).Count);
assertEquals(QTY, repo.getById(cart.id()).getQuantity(ITEM_NAME));
```

intelliware.ca
software development

Citations / Resources

- why we need another mocking framework?
- Mockito documentation and examples
- TotT: Friends You Can Depend On

intelliware.ca
software development