Table of Contents (left)

RSpec Mocks

rspec-mocks is a test-double framework for rspec with support for method stubs, fakes, and message expectations on generated test-doubles and real objects alike.

Install

```
gem install rspec       # for rspec-core, rspec-expectations, rspec-mocks
gem install rspec-mocks # for rspec-mocks only
```

Test Doubles

A Test Double is an object that stands in for a real object in a test. RSpec creates test doubles that support method stubs and message expectations.

```
book = double("book")
```

Method Stubs

A method stub is an implementation that returns a pre-determined value. Method stubs can be declared on test doubles or real objects using the same syntax. rspec-mocks supports 3 forms for declaring method stubs:

```
book.stub(:title) { "The RSpec Book" }
book.stub(:title => "The RSpec Book")
book.stub(:title).and_return("The RSpec Book")
```

You can also use this shortcut, which creates a test double and declares a method stub in one statement:

```
book = double("book", :title => "The RSpec Book")
```

The first argment is a name, which is used for documentation and appears in failure messages. If you don't care about the name, you can leave it out, making the combined instantiation/stub declaration very terse:

```
double(:foo => 'bar')
```

This is particularly nice when providing a list of test doubles to a method that iterates through them:

```
order.calculate_total_price(stub(:price => 1.99),stub(:price => 2.99))
```

Consecutive return values

When a stub might be invoked more than once, you can provide additional arguments to `and_return`. The invocations cycle through the list. The last value is returned for any subsequent invocations:

```
die.stub(:roll).and_return(1,2,3)
die.roll # => 1
die.roll # => 2
die.roll # => 3
die.roll # => 3
die.roll # => 3
```

To return an array in a single invocation, declare an array:

```
team.stub(:players).and_return([stub(:name => "David")])
```

Message Expectations

A message expectation is an expectation that the test double will receive a message some time before the example ends. If the message is received, the expectation is satisfied. If not, the example fails.

```
validator = double("validator")
validator.should_receive(:validate).with("02134")
zipcode = Zipcode.new("02134", validator)
zipcode.valid?
```

Nomenclature

Mock Objects and Test Stubs

The names Mock Object and Test Stub suggest specialized Test Doubles. i.e. a Test Stub is a Test Double that only supports method stubs, and a Mock Object is a Test Double that supports message expectations and method stubs.

There is a lot of overlapping nomenclature here, and there are many variations of these patterns (fakes, spies, etc). Keep in mind that most of the time we're talking about method-level concepts that are variations of method stubs and message expectations, and we're applying to them to *one* generic kind of object: a Test Double.

Test-Specific Extension

a.k.a. Partial Stub/Mock, a Test-Specific Extension is an extension of a real object in a system that is instrumented with test-double like behaviour in the context of a test. This technique is very common in Ruby because we often see class objects acting as global namespaces for methods. For example, in Rails:

```
person = double("person")
Person.stub(:find) { person }
```

In this case we're instrumenting Person to return the person object we've defined whenever it receives the `find` message. We can do this with any object in a system because rspec-mocks adds the `stub` and `should_receive` methods to every object. When we use either, RSpec replaces the method we're stubbing or mocking with its own test-double-like method. At the end of the example, RSpec verifies any message expectations, and then restores the original methods.

Expecting Arguments

```
double.should_receive(:msg).with(*args)
double.should_not_receive(:msg).with(*args)
```

Argument Matchers

Arguments that are passed to `with` are compared with actual arguments received using ==. In cases in which you want to specify things about the arguments rather than the arguments themselves, you can use any of the matchers that ship with rspec-expectations. They don't all make syntactic sense (they were primarily designed for use with RSpec::Expectations), but you are free to create your own custom RSpec::Matchers.

rspec-mocks also adds some keyword Symbols that you can use to specify certain kinds of arguments:

```
double.should_receive(:msg).with(no_args())
double.should_receive(:msg).with(any_args())
double.should_receive(:msg).with(1, kind_of(Numeric), "b") #2nd argument can any kind of Numer
double.should_receive(:msg).with(1, boolean(), "b") #2nd argument can true or false
double.should_receive(:msg).with(1, /abc/, "b") #2nd argument can be any String matching the s
double.should_receive(:msg).with(1, anything(), "b") #2nd argument can be anything at all
double.should_receive(:msg).with(1, ducktype(:abs, :div), "b")
                    #2nd argument can be object that responds to #abs and #div
```

Receive Counts

```
double.should_receive(:msg).once
double.should_receive(:msg).twice
double.should_receive(:msg).exactly(n).times
double.should_receive(:msg).at_least(:once)
double.should_receive(:msg).at_least(:twice)
double.should_receive(:msg).at_least(n).times
double.should_receive(:msg).at_most(:once)
double.should_receive(:msg).at_most(:twice)
double.should_receive(:msg).at_most(n).times
double.should_receive(:msg).any_number_of_times
```

Ordering

```
double.should_receive(:msg).ordered
double.should_receive(:other_msg).ordered
  #This will fail if the messages are received out of order
```

Setting Reponses

Whether you are setting a message expectation or a method stub, you can tell the object precisely how to respond. The most generic way is to pass a block to `stub` or `should_receive`:

```
double.should_receive(:msg) { value }
```

When the double receives the `msg` message, it evaluates the block and returns the result.

```
double.should_receive(:msg).and_return(value)
double.should_receive(:msg).exactly(3).times.and_return(value1, value2, value3)
  # returns value1 the first time, value2 the second, etc
double.should_receive(:msg).and_raise(error)
```

```
  #error can be an instantiated object or a class
  #if it is a class, it must be instantiable with no args
double.should_receive(:msg).and_throw(:msg)
double.should_receive(:msg).and_yield(values,to,yield)
double.should_receive(:msg).and_yield(values,to,yield).and_yield(some,other,values,this,time)
  # for methods that yield to a block multiple times
```

Any of these responses can be applied to a stub as well

```
double.stub(:msg).and_return(value)
double.stub(:msg).and_return(value1, value2, value3)
double.stub(:msg).and_raise(error)
double.stub(:msg).and_throw(:msg)
double.stub(:msg).and_yield(values,to,yield)
double.stub(:msg).and_yield(values,to,yield).and_yield(some,other,values,this,time)
```

Arbitrary Handling

Once in a while you'll find that the available expectations don't solve the particular problem you are trying to solve. Imagine that you expect the message to come with an Array argument that has a specific length, but you don't care what is in it. You could do this:

```
double.should_receive(:msg) do |arg|
  arg.size.should eq(7)
end
```

Combining Expectation Details

Combining the message name with specific arguments, receive counts and responses you can get quite a bit of detail in your expectations:

```
double.should_receive(:<<).with("illegal value").once.and_raise(ArgumentError)
```

While this is a good thing when you really need it, you probably don't really need it! Take care to specify only the things that matter to the behavior of your code.

Further Reading

There are many different viewpoints about the meaning of mocks and stubs. If you are interested in learning more, here is some recommended reading:

- Mock Objects: http://www.mockobjects.com/
- Endo-Testing: http://www.mockobjects.com/files/endotesting.pdf
- Mock Roles, Not Objects: http://www.mockobjects.com/files/mockrolesnotobjects.pdf
- Test Double Patterns: http://xunitpatterns.com/Test%20Double%20Patterns.html
- Mocks aren't stubs: http://www.martinfowler.com/articles/mocksArentStubs.html

Also see

- http://github.com/rspec/rspec
- http://github.com/rspec/rspec-core
- http://github.com/rspec/rspec-expectations

Generated on Wed Jan 4 23:42:14 2012 by yard 0.7.4 (ruby-1.9.3).