Rhino Mocks

Petr Onderka

- mock object framework for .Net
- goal: simplification of unit testing
- allows creating mock implementations of objects and verifying interactions
- used in conjunction with a unit testing framework, like NUnint

History

- version 1.0 released in 2005, for .Net 1.1
- current version is 3.6, released in 2009 for .Net 3.5
- work on 4.0 is ongoing, under a new developer
  - primary goal is to remove historical baggage from the interface

Links

- Original repository (3.6): https://github.com/ayende/rhino-mocks
- New repository (4.0): https://github.com/meisinger/rhino-mocks
- Wiki:
  - http://www.ayende.com/Wiki/Rhino+Mocks.ashx
  - http://www.ayende.com/Wiki/Rhino+Mocks+3.5.ashx

Mock implementation

- class that we want to test has a dependency
- we can't or don't want to use production version of that dependency
- creating a full implementation manually is too cumbersome
- solution: use Rhino Mocks to create a mock implementation, implementing only the members that we need, using a few lines of code

Mock implementation example

```
var foo = MockRepository.GenerateStub<IFoo>();
foo.Stub(x => x.GetValue(42)).Return("forty two");

// in real test, this would call the tested method
var value = foo.GetValue(42);

Assert.AreEqual("forty two", value);
```

Interaction verification

- we want to verify that the tested class calls a certain method, with the right arguments
- possibly also verify that the method is called given number of times

Interaction verification example

```
var foo = MockRepository.GenerateStrictMock<IFoo>();
foo.Expect(x => x.GetValue(42)).Return("forty two");


var value = foo.GetValue(42);


Assert.AreEqual("forty two", value);
foo.VerifyAllExpectations();
```

Mock objects kinds

- various kinds of objects:
  - dynamic mock: un-mocked members return the default value (null, 0, false)
  - strict mock: un-mocked members throw
  - partial mock: un-mocked members call base class
  - stub: similar to dynamic mock, except un-mocked properties and events behave normally
- stubs are used to provide implementation, mocks to verify expectations

Arguments

- various ways to configure what arguments are accepted for a mocked member:
  - default: same as in the Stub or Expect lambda
  - append .IgnoreArguments() to accept any arguments
  - use Arg<T>.Matches(lambda) to check that arguments fulfil some condition

Arguments examples

```
foo.Expect(x => x.GetValue(0))
    .IgnoreArguments()
    .Return("some number");

foo.Expect(
     x => x.GetValue(Arg<int>.Matches(i => i % 2 == 0)))
    .Return("even number");
```

Technical details

- needs to dynamically create a new class that implements interface or inherits from class at runtime
- uses Castle.DynamicProxy for that
  - library that creates proxy classes, allows intercepting calls to their members
- this means that only interfaces and virtual members of classes can be mocked

The End

Questions?