will set up expectations that method b() be called in the mock. If b() is not called, a runtime error is generated.

Multiple ExpectCall methods may be called on the same mock. This sets up expectation that each of those methods will be called in order. For instance, the following sets up expecation that method b() will be called, and then method c(), in that order.

```
mocks.ExpectCall(barMock, IBar::b);
mocks.ExpectCall(barMock, IBar::c);
```

If order doesn't matter, autoExpect can be called prior to calls to ExpectCall

```
mocks.autoExpect = false;
```

Method returns a TCall object, which can be used to further refine the expectations of the method call.

OnCall() method

```
TCall MockRepository::OnCall( mock, method )
```

Similar to ExpectCall, except that it's okay if the method doesn't get called.

NeverCall() method

```
TCall MockRepository::NeverCall( mock, method )
```

Similar to ExpectCall, except that it sets up expectation that the method will never be called.

ExpectCallOverload() method

```
TCall MockRepository::ExpectCallOverload( mock, method )
```

If IBar has two methods named c() with different overloads:

```
virtual int c(std::string) = 0;
virtual int c(int) = 0;
```

The typical "Expect" call won't work:

```
mocks.ExpectCall(barMock, IBar::c);
```

ExpectCallOverload must be used:

```
mocks.ExpectCallOverload(barMock, (int(IBar::*)(std::string))&IBar::c);
```

Similarly to how ExpectCallOverload is the overload version of ExpectCall:

- OnCallOverload is the overload version of OnCall
- NeverCallOverload is the overload version of NeverCall

TCall class

This class is not instantiated directly. It's returned by MockRepository::ExpectCall.The TCall class can be used to further refine the expectations of the method call.

With() method

When used, indicates what parameters expected to be passed in.

For instance, the following call sets up expectation that method c will be called with argument "hello"

```
mocks.ExpectCall(barMock, IBar::c).With("hello");
```

With returns the TCall object back, so further refinements can be chained. See TCall::Return

Return() method

```
TCall::Return
```

When used, indicates what value the method will return.

For instance, the following call sets up expectation that method c will return 42.

```
mocks.ExpectCall(barMock, IBar::c).Return(42);
```

Since TCall::With returns the TCall object, With and Return can be used in the same statement. The following call sets up expectation that method c will return 42 when it's passed argument "hello":

```
mocks.ExpectCall(barMock, IBar::c).With("hello").Return(42);
```

With returns the TCall object back, so further refinements can be chained. See TCall::Return

Throw() method

```
TCall::Throw
```

When used, indicates that the method must throw an exception.

For instance, the following call sets up expectation that method c will throw an exception when argument "fail" is passed in.

```
mocks.ExpectCall(barMock, IBar::c).With("fail").Throw(std::exception());
```

With returns the TCall object back, so further refinements can be chained. See TCall::Return

Do() method

```
TCall::Do
```

When used, indicates what the method will do by passing in a function pointer.

For instance, the following is another way to set up expectation that when argument "hello" is passed in, method c() returns 99:

```
mocks.ExpectCall(barMock,
IBar::c).With("hello").Do(SomeGlobalFunction);
```

where SomeGlobalFunction is some static or global function defined as:

```
int someOtherFunction(std::string s)
{
    if ( s == "hello" )
```

```
}
```

With returns the TCall object back, so further refinements can be chained. See TCall::Return

After() method

```
TCall::After( Call& )
```

The After method is used to specify that a particular Expect call should come after the another Expect call.

The following sets up the expectation for method c() as follows:

- if c() is called with "hello", 1 is returned.
- if c() is called with "world", 2 is returned.
- if c() is called with "1", 3 is returned.
- if c() is called with "2", 4 is returned.
- The order of the calls don't matter, because of autoExpect = false, except:
  - call with argument "1" must come after call with argument "hello" and
  - call with argument "2" must come after call with argument "world"

```
mocks.autoExpect = false;
Call &callOne = mocks.ExpectCall(barMock,
IBar::c).With("hello").Return(1);
Call &callTwo = mocks.ExpectCall(barMock,
IBar::c).With("world").Return(2);
mocks.ExpectCall(barMock, IBar::c).After(callOne).With("1").Return(3);
mocks.ExpectCall(barMock, IBar::c).After(callTwo).With("2").Return(4);
```

With returns the TCall object back, so further refinements can be chained. See TCall::Return

Categories

1 comment

A Fandom user • 1/11/2021

Hi,
Thanks for writing this article. It is precise and easy to understand. I usually struggle when going through manuals and implementing their functionalities. But your article was an exception. It helped me to understand and grasp the concepts very quickly. Thanks again. :)