Mock Objects in Xojo

Yves Meynard
Software Developer
Lightspeed

Why Mock Objects?

- For unit tests
- When there are expensive dependencies
- We know the requests that will be made and what responses should be returned
- Example: an invoice object reads its info from a database. Create, fill then destroy a database, just for one test?

Why Mock Objects? (2)

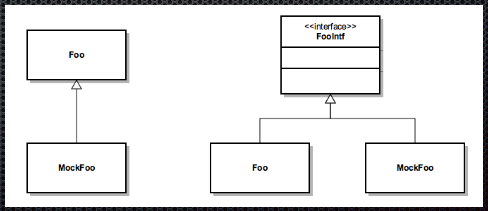
- Don't make expensive requests to the real object, talk to the mock object
- The mock has been told what to expect, and what to reply — we can simulate errors at will
- Test what requests the object makes as well as its handling of all possible responses

What is a Mock Object?

- Inexpensive to construct
- Calling code must not see a difference
- The mock object knows what calls it may receive
- For each call, it knows the response it must give
- An unexpected call signals an error
- We can ask if all expected calls were made

How to code one?

- Must have the same interface as the object it mocks
- In Xojo, it must be a subclass of the mocked class
- Unless they can both implement the same Interface



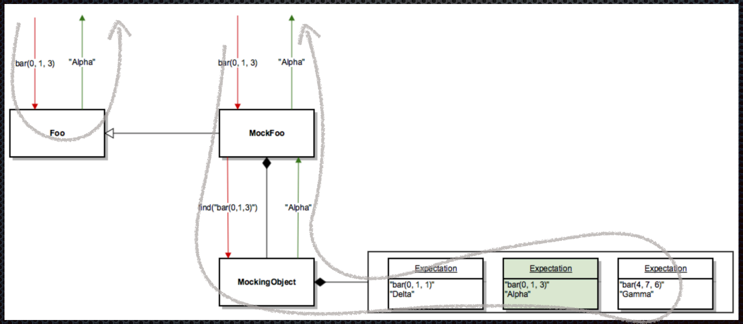
How do we substitute the MockObject?

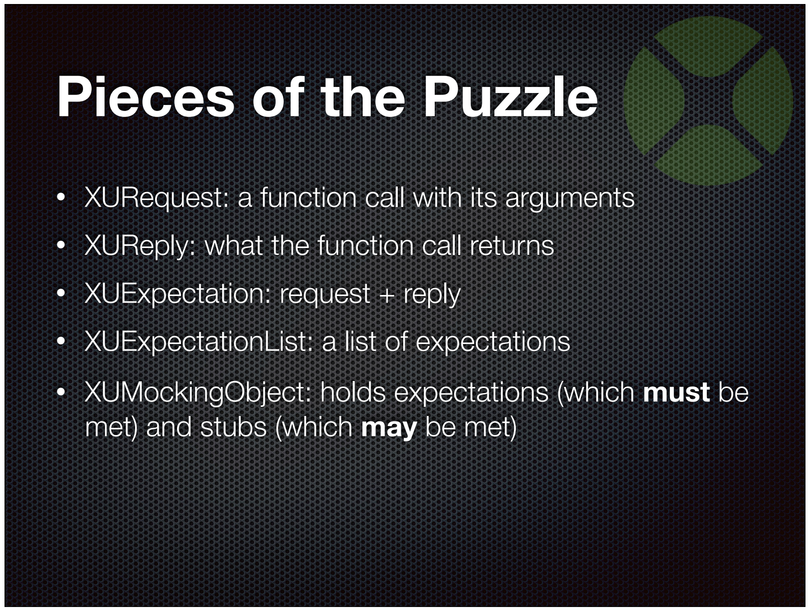
- "Dependency Injection"
  - In the constructor: `new Thing(mockFoo)`
    Most verbose approach
  - In a setter: `thing->setFoo(mockFoo)`
    Must define a new accessor
- Using global state: `#if BuildConstants.kIsUnitTestBuild`
  Not an injection, only works for Singletons

Generic Mock Objects

- In Lightspeed OnSite, we intercept calls to LSSDatabase and send them to LSDatabaseMock
- Convenient, but ugly and clumsy
- Works because LSSDatabase is a singleton
- Better approach: factor out the mocking logic
- Mock **any** class by dropping in a mocking object as a property

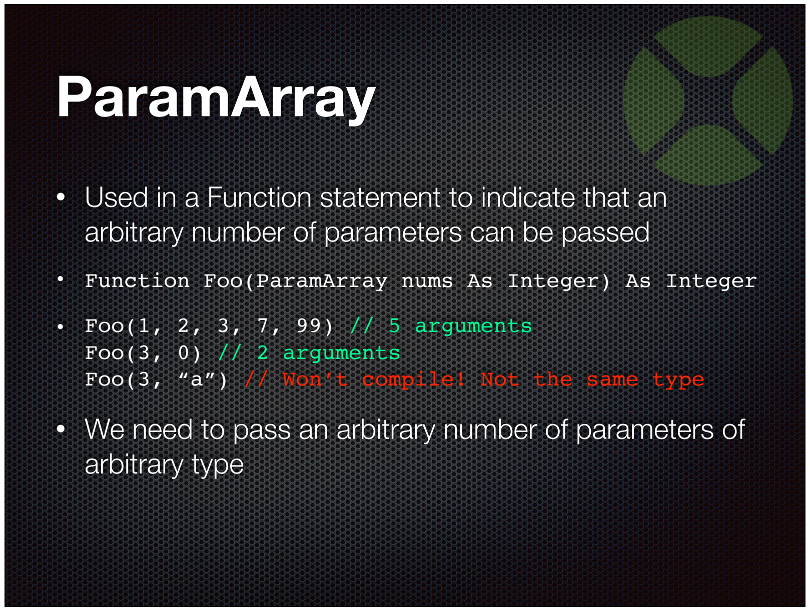
Logic Flow





Pieces of the Puzzle

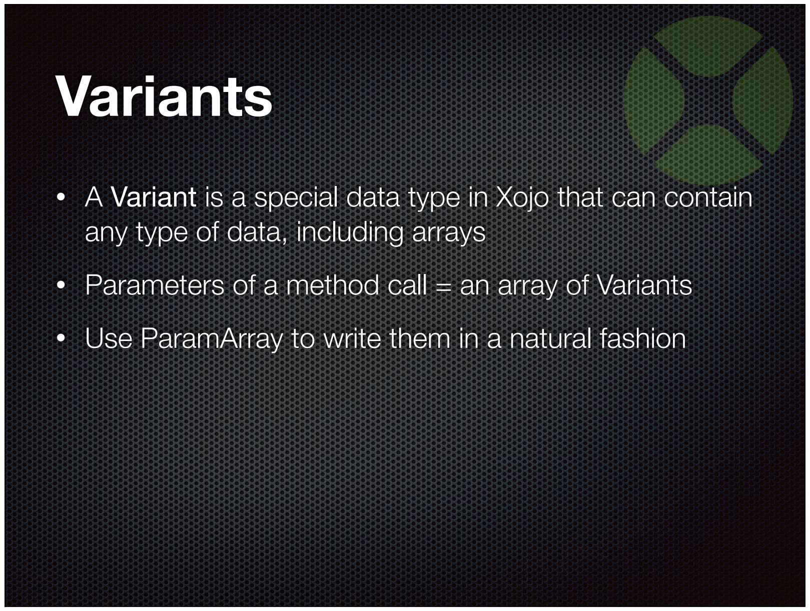
- XURequest: a function call with its arguments
- XUReply: what the function call returns
- XUExpectation: request + reply
- XUExpectationList: a list of expectations
- XUMockingObject: holds expectations (which **must** be met) and stubs (which **may** be met)



ParamArray

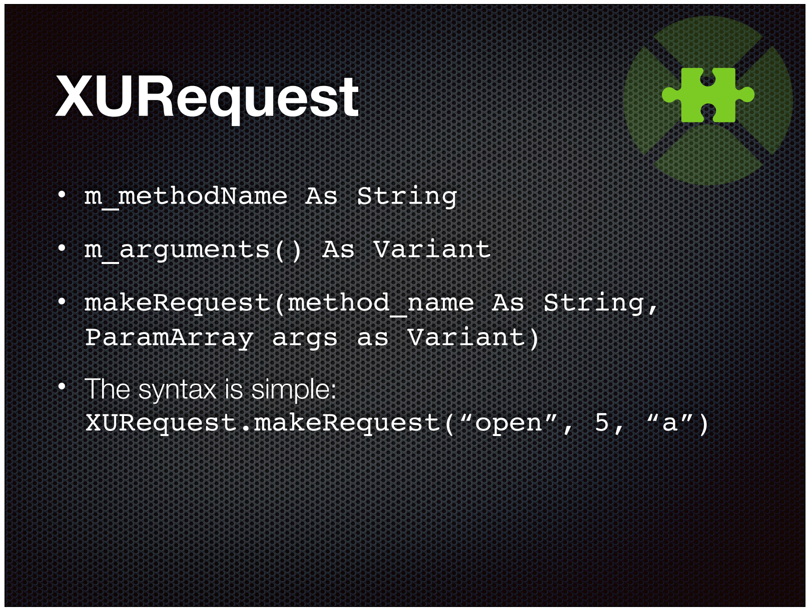
- Used in a Function statement to indicate that an arbitrary number of parameters can be passed
- `Function Foo(ParamArray nums As Integer) As Integer`

-
  ```
  Foo(1, 2, 3, 7, 99) // 5 arguments
  Foo(3, 0) // 2 arguments
  Foo(3, “a”) // Won't compile! Not the same type
  ```
- We need to pass an arbitrary number of parameters of arbitrary type



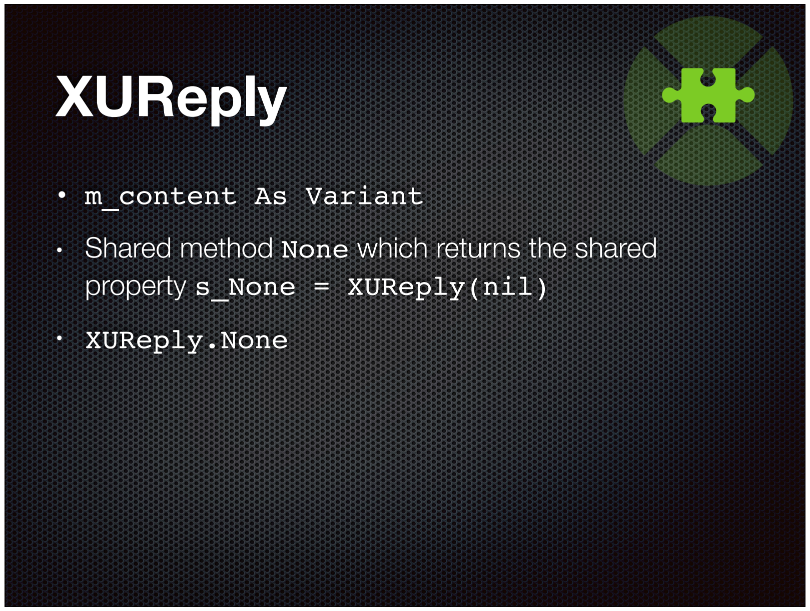
Variants

- A Variant is a special data type in Xojo that can contain any type of data, including arrays
- Parameters of a method call = an array of Variants
- Use ParamArray to write them in a natural fashion



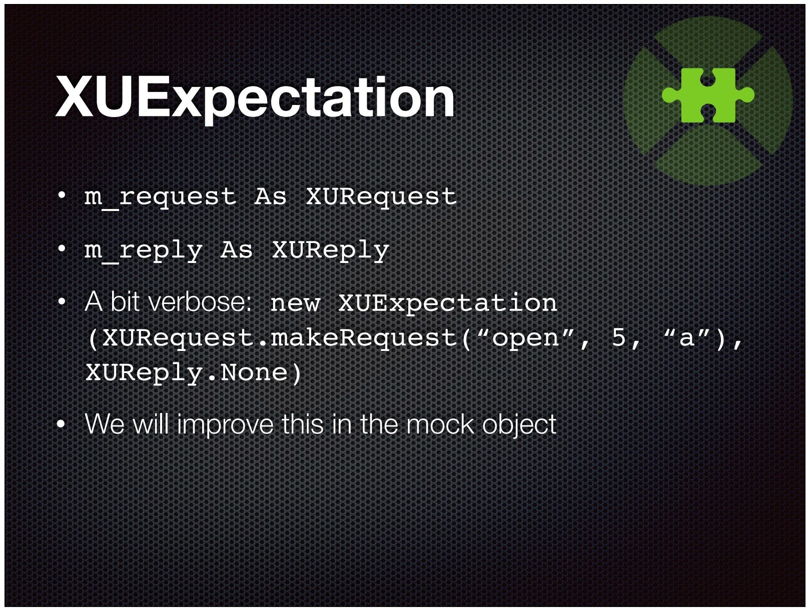
XURequest

- `m_methodName As String`
- `m_arguments() As Variant`
- `makeRequest(method_name As String, ParamArray args as Variant)`
- The syntax is simple:
  `XURequest.makeRequest(“open”, 5, “a”)`



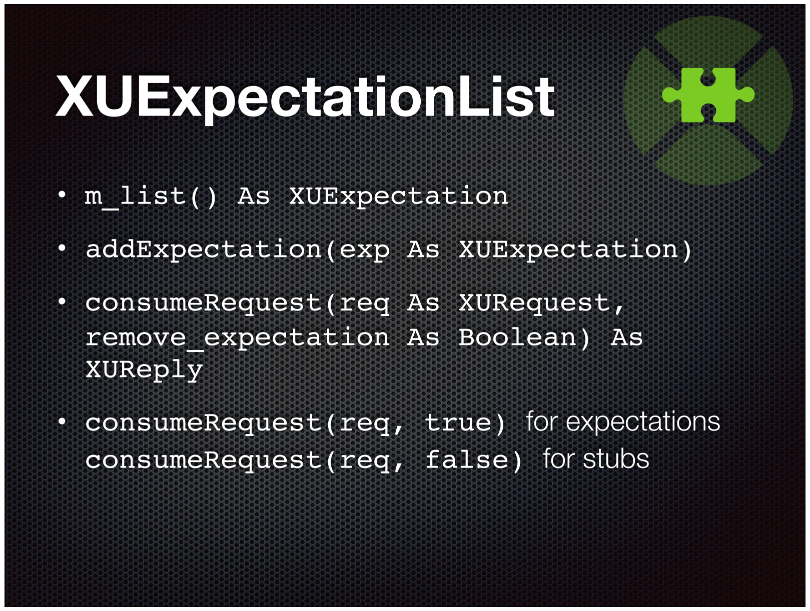
XUReply

- `m_content As Variant`
- Shared method `None` which returns the shared property `s_None = XUReply(nil)`
- `XUReply.None`



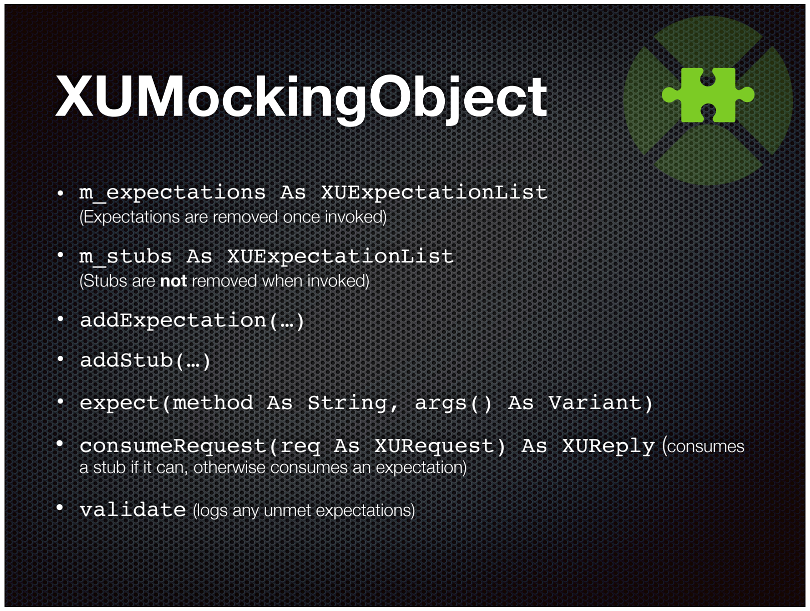
XUExpectation

- `m_request As XURequest`
- `m_reply As XUReply`
- A bit verbose: `new XUExpectation (XURequest.makeRequest(“open”, 5, “a”), XUReply.None)`
- We will improve this in the mock object



XUExpectationList

- `m_list() As XUExpectation`
- `addExpectation(exp As XUExpectation)`
- `consumeRequest(req As XURequest, remove_expectation As Boolean) As XUReply`
- `consumeRequest(req, true)` for expectations
  `consumeRequest(req, false)` for stubs



XUMockingObject

- `m_expectations As XUExpectationList`
  (Expectations are removed once invoked)
- `m_stubs As XUExpectationList`
  (Stubs are **not** removed when invoked)
- `addExpectation(…)`
- `addStub(…)`
- `expect(method As String, args() As Variant)`
- `consumeRequest(req As XURequest) As XUReply` (consumes a stub if it can, otherwise consumes an expectation)
- `validate` (logs any unmet expectations)

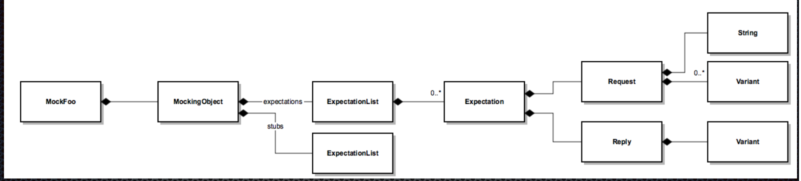
Mocking an Object

- Add a MockingObject as a property
- Intercept calls: ask the mocker to consume them instead, and return its reply
- `Foo.open(id As Integer, s As String) As Integer`
-
```
MockFoo.open(id As Integer, s As String) As Integer
req = makeRequest(“open”, id, s)
Dim reply As XUReply = m_mocker.consumeRequest(req)
return reply.m_content.integerValue()
```

Mocking an Object (2)

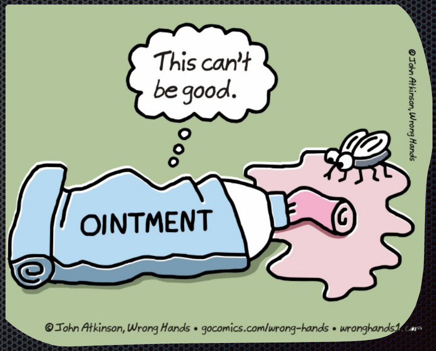
- Define methods to create Expectations in MockFoo
-
```
expect(name As String, ParamArray args As Variant)
    Dim request As new Request(name, args)
    mocker.addExpectation(request, Reply.None)
```
-
```
expect_draw(x As Double, y As Double, s As String)
    self.expect(“draw”, x, y, z)
```
-
```
mock.expect_draw(0.0, -5.0, “Bottom”)
mock.expect_draw(5.0, 0.0, “Right”)
mock.expect_draw(0.0, 5.0, “Top”)
mock.expect_draw(-5.0, 0.0, “Left”)
```

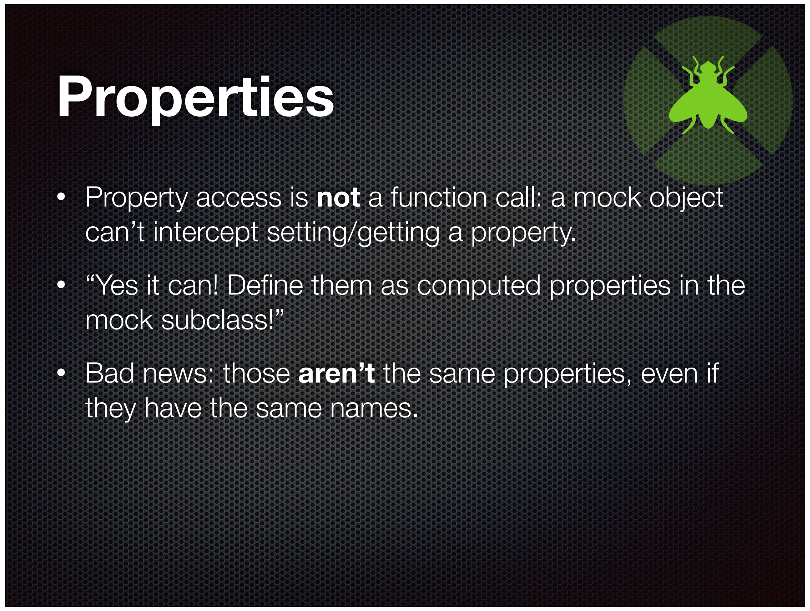
Our Classes



Flies in the Ointment

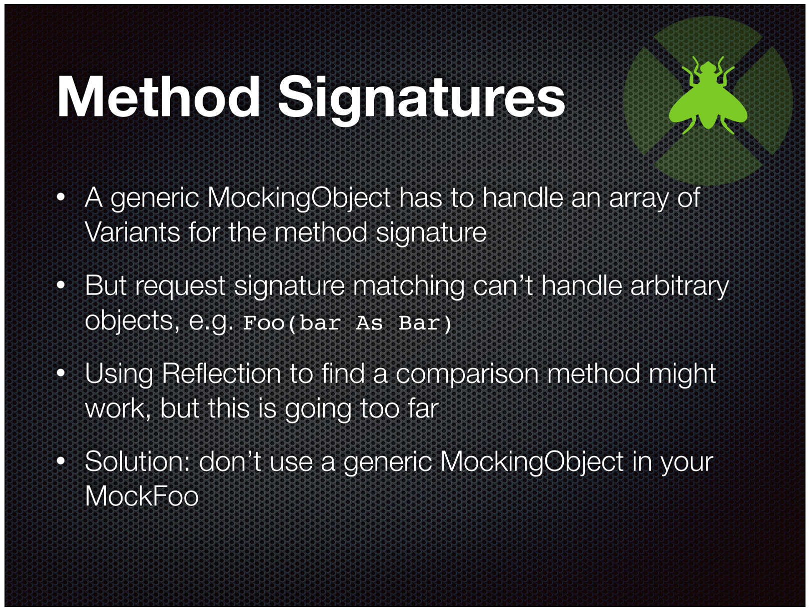
- Properties
- Object Parameters
- Operator_compare


This can't be good.
OINTMENT
© John Atkinson, Wrong Hands • gocomics.com/wrong-hands • wronghands1.com
© John Atkinson, Wrong Hands



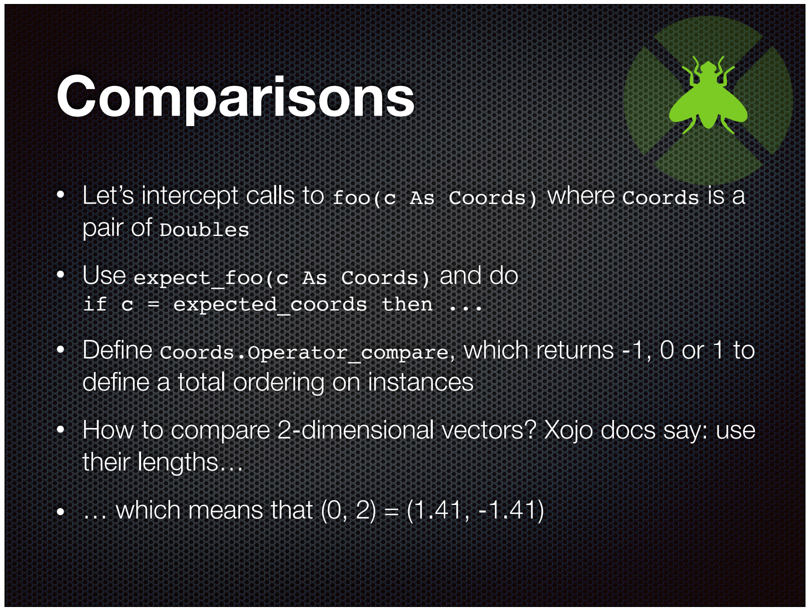
Properties

- Property access is **not** a function call: a mock object can't intercept setting/getting a property.
- "Yes it can! Define them as computed properties in the mock subclass!"
- Bad news: those **aren't** the same properties, even if they have the same names.



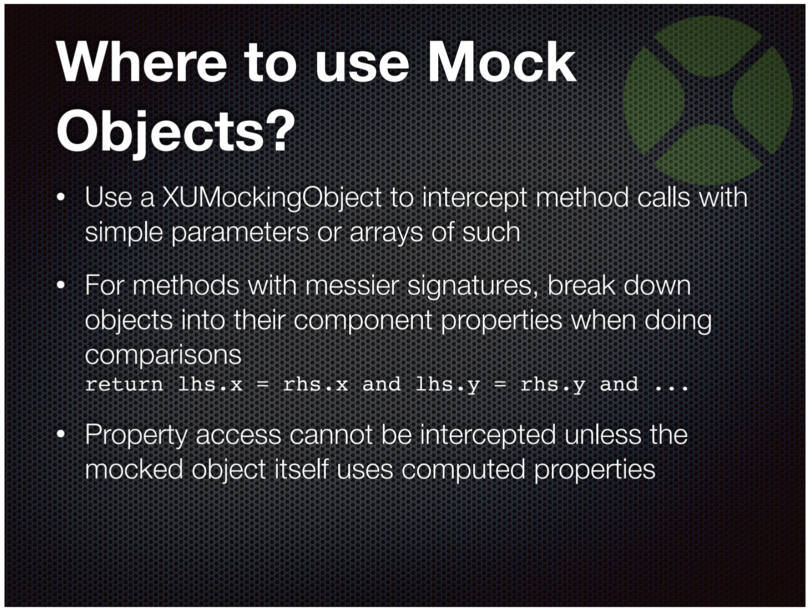
Method Signatures

- A generic MockingObject has to handle an array of Variants for the method signature
- But request signature matching can't handle arbitrary objects, e.g. `Foo(bar As Bar)`
- Using Reflection to find a comparison method might work, but this is going too far
- Solution: don't use a generic MockingObject in your MockFoo



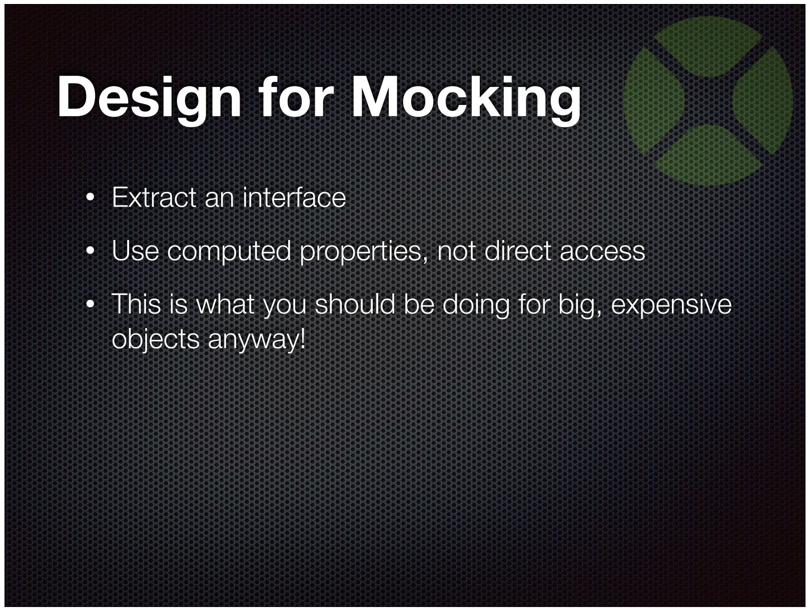
Comparisons

- Let's intercept calls to `foo(c As Coords)` where `Coords` is a pair of `Doubles`
- Use `expect_foo(c As Coords)` and do
  `if c = expected_coords then ...`
- Define `Coords.Operator_compare`, which returns -1, 0 or 1 to define a total ordering on instances
- How to compare 2-dimensional vectors? Xojo docs say: use their lengths…
- … which means that (0, 2) = (1.41, -1.41)



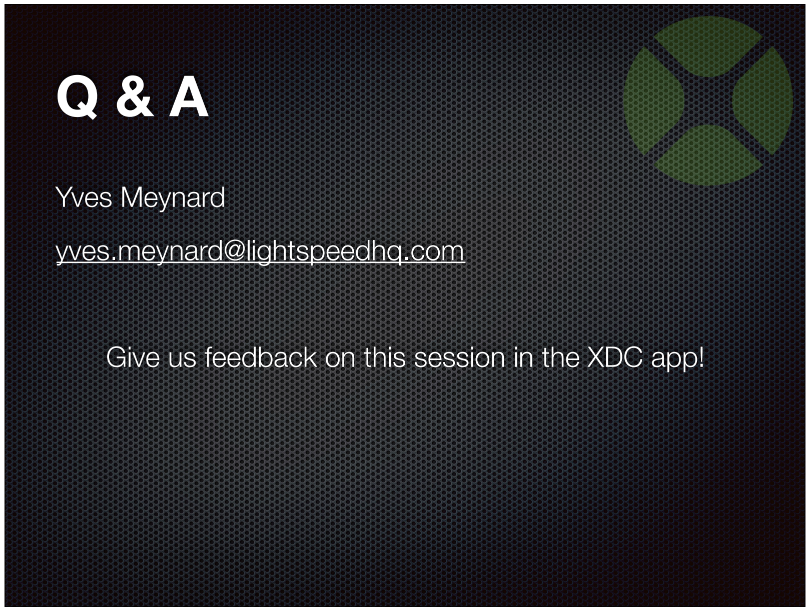
Where to use Mock Objects?

- Use a XUMockingObject to intercept method calls with simple parameters or arrays of such
- For methods with messier signatures, break down objects into their component properties when doing comparisons
  `return lhs.x = rhs.x and lhs.y = rhs.y and ...`
- Property access cannot be intercepted unless the mocked object itself uses computed properties



Design for Mocking

- Extract an interface
- Use computed properties, not direct access
- This is what you should be doing for big, expensive objects anyway!



Q & A

Yves Meynard

yves.meynard@lightspeedhq.com

Give us feedback on this session in the XDC app!