A Mocking Framework: Laugh at Unit Tests

André-Claude Gendron
CIUSSS de l'Estrie – CHUS

Andreas Dieckow
InterSystems Corporation

Guest Speaker

CIUSSS de l'Estrie – CHUS

- Centre Intégré Universitaire de Santé et de Services Sociaux de l'Estrie - Centre Hospitalier Universitaire de Sherbrooke
- More than 17,000 employees
- Over 1,000 physicians
- Provides healthcare to more than 500,000 people.

André-Claude

- Graduated in 2007 in computer engineering
- Working at CHUS since 2012 in the R&D dept
- Introduced source control, peer review, automatic tests & agile to management
- Team wanted to do unit tests using mocks and couldn't. So decided to create it !

A MOCKING FRAMEWORK: LAUGH AT UNIT TESTS

André-Claude Gendron
CIUSSS de l'Estrie – CHUS, Canada









TOGETHER for LIFE

Québec

PRESENTATION OBJECTIVES

- **Extend the Eclipse Unit Test framework to simulate the behavior of a real method/object.**

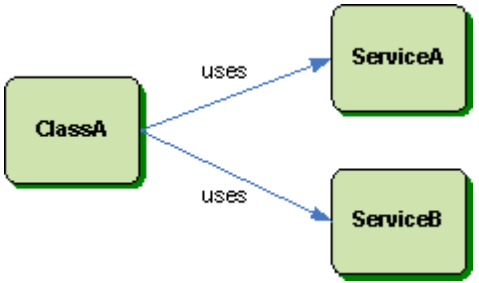
- Give an overview of how a Mocking framework will help and how it works.
- Show the power of using a Mocking framework by giving an extensive demonstration.
- Code will be made available in InterSystems Developer Community.

PROGRAMMING BACKGROUND
UNIT TESTS

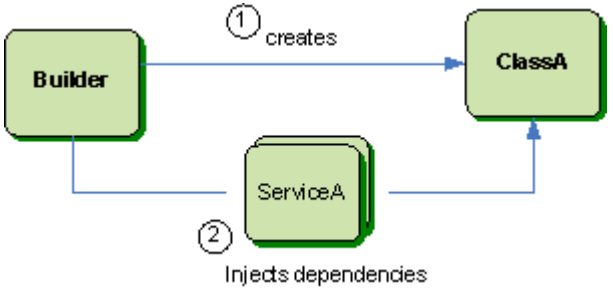
- Code used to validate proper behavior of another object or function
  - Shows that the individual parts are correct.
  - Provides a strict, written contract that the piece of code must satisfy.
- Helps
  - Find problems early
  - Deal with change

PROGRAMMING BACKGROUND
INVERSION OF CONTROL DEPENDENCY INJECTION

- Tight Coupling



- Dependency Injection Pattern



Source : https://gallery.technet.microsoft.com/Inversion-of-Control-with-b7ed6976

PROGRAMMING BACKGROUND
MOCK OBJECTS

- “In object-oriented programming, mock objects are simulated objects that mimic the behavior of real objects in controlled ways”
- Expects to receive calls with :
  - Precise sequence
  - Specified parameters
- Returns precise output

Source : https://en.wikipedia.org/wiki/Mock_object

PROGRAMMING BACKGROUND
MOCK OBJECTS ARE NOT STUBS

STUBS

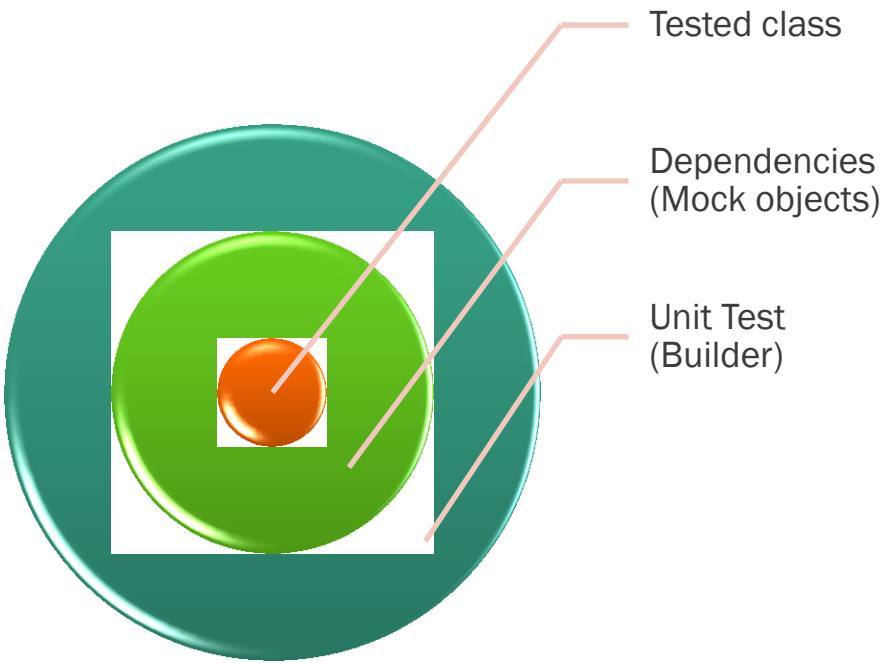
- Make sure tested class gives the right result when fed some data

MOCK OBJECTS

- Make sure tested class was contacted in exactly the right way

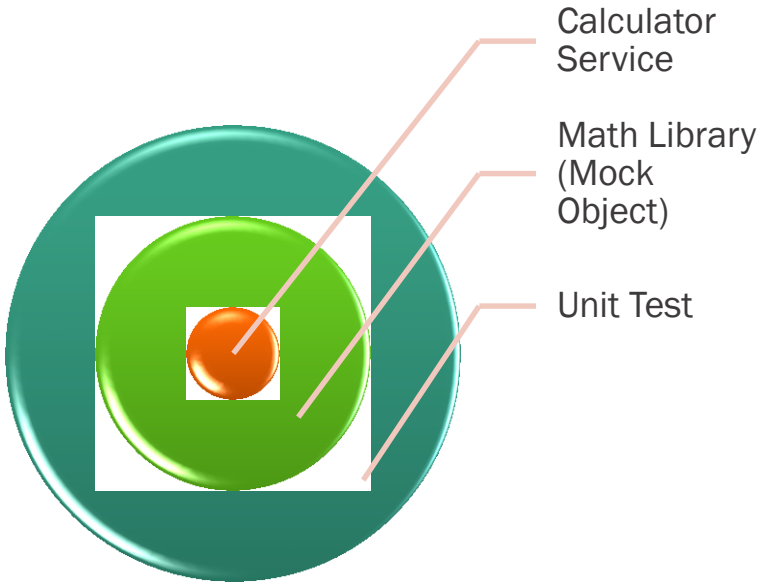
Source : https://en.wikipedia.org/wiki/Mock_object

THE IDEA : ISOLATION

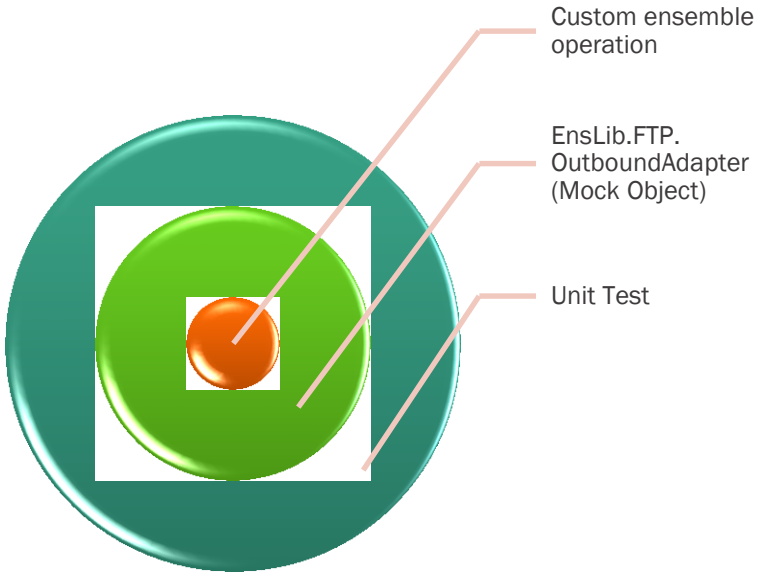


SOME EXAMPLES

- Calculator Service using some math library

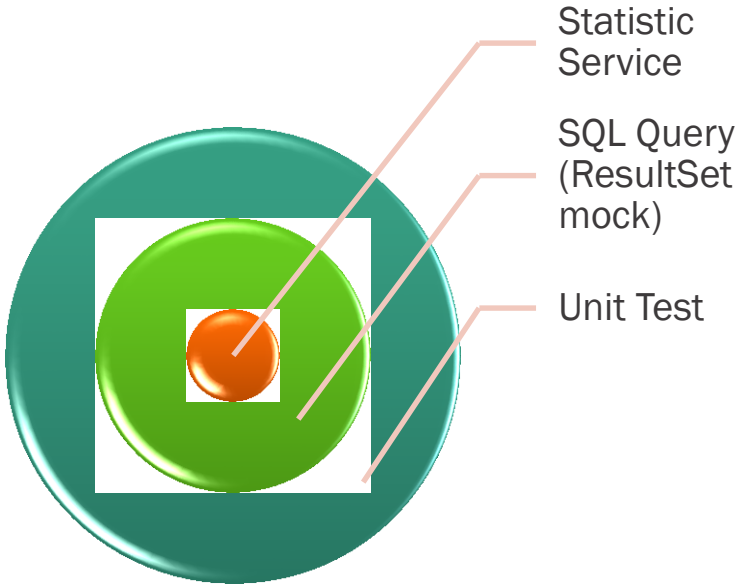


- A custom Ensemble Operation using a FTP Outbound adaptor

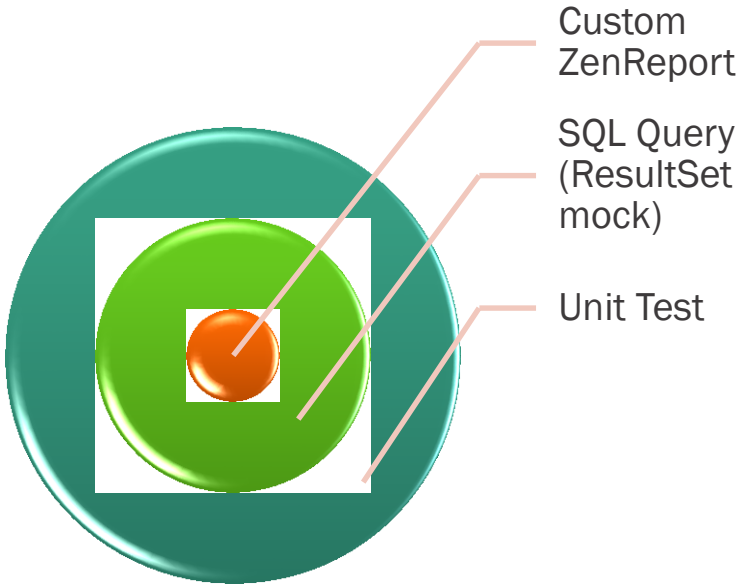


SOME EXAMPLES

- Some Statistic Service using a SQL Query



- ZenReport using a SQL Query



ANOTHER REPRESENTATION OF MOCKS

REAL SYSTEM

Green = class in focus
Yellow = dependencies
Grey = other unrelated classes

CLASS IN UNIT TEST

Green = class in focus
Yellow = mocks for the unit test

Source : https://zeroturnaround.com/

A SIMPLE “REAL-WORLD” EXAMPLE

CTestedClassWithoutInjection.cls

```
Class MockDemo.CTestedClassWithoutInjection Extends %RegisteredObject
{

Method SumOperation(A As %Integer, B As %Integer) As %Integer
{
    set mathService = ##class(MathService).%New()
    quit mathService.Add(A, B)
}

}
```

MathService.cls

```
Class MockDemo.MathService Extends %RegisteredObject
{

Method Add(A As %Integer, B As %Integer) As %Integer
{
    quit A+B
}

/// TODO : Implement division some day

}
```

CTestedClassWithoutInjection.cls | CTestTestedClassWithoutInjection.cls

```
Class MockDemo.UnitTests.CTestTestedClassWithoutInjection Extends Tests.Fw.CUnitTestBase
{

// --- Run test ---

Property testedClass As MockDemo.CTestedClassWithoutInjection [ Private ];

ClassMethod RunTests()
{
    do ##super()
}

Method OnBeforeOneTest(testname As %String) As %Status
{
    set ..testedClass = ##class(MockDemo.CTestedClassWithoutInjection).%New()
    quit $$$OK
}

// --- Tests for ComplexSumOperation ---

Method TestSumOperation()
{
    do $$$AssertEquals(..testedClass.SumOperation(1, 2), 3)
}

}
```

```
==========================================================================
Suite: MockDemo.UnitTests.CTestTestedClassWithoutInjection
==========================================================================
  MockDemo.UnitTests.CTestTestedClassWithoutInjection begins ...
    MockDemo.UnitTests.CTestTestedClassWithoutInjection begins ...
      TestSumOperation() begins ...
        AssertEquals:: 3==3 in ..testedClass.SumOperation(1, 2)== 3 (passed)
        LogMessage:Duration of execution: .001031 sec.
      TestSumOperation passed
    MockDemo.UnitTests.CTestTestedClassWithoutInjection passed
  MockDemo.UnitTests.CTestTestedClassWithoutInjection passed
```

MathService.cls

```
Class MockDemo.MathService Extends %RegisteredObject
{

Method Add(A As %Integer, B As %Integer) As %Integer
{
    quit A+B+1
}

/// TODO : Implement division some day

}
```

```
================================================================================
Suite: MockDemo.UnitTests.CTestTestedClassWithoutInjection
================================================================================
  MockDemo.UnitTests.CTestTestedClassWithoutInjection begins ...
    MockDemo.UnitTests.CTestTestedClassWithoutInjection begins ...
      TestSumOperation() begins ...
AssertEquals:: 4==3 in
MockDemo.UnitTests.C
mOperation:
        LogMessage:Durat
      TestSumOperation f
    MockDemo.UnitTests
  MockDemo.UnitTests.C
```

CTestedClassWithoutInjection.cls

```
Class MockDemo.CTestedClassWithoutInjection Extends %RegisteredObject
{

Method SumOperation(A As %Integer, B As %Integer) As %Integer
{
    set mathService = ##class(MathService).%New()
    quit mathService.Add(A, B)
}

}
```

THE SAME EXAMPLE, USING THE MOCK OBJECT

MathService.cls | CTestedClass.cls

```
Class MockDemo.CTestedClass Extends %RegisteredObject
{

Property mathService As MathService [Private];

Method %OnNew(mathService As MathService
{
    set ..mathService = mathService
    quit $$$OK
}

Method SumOperation(A As %Integer, B As %
{
    quit ..mathService.Add(A, B)
}
}
```

CTestedClassWithoutInjection.cls

```
Class MockDemo.CTestedClassWithoutInjection E
{

Method SumOperation(A As %Integer, B As %Inte
{
    set mathService = ##class(MathService).%Ne
    quit mathService.Add(A, B)
}

}
```

HOW TO USE IT

- MockManager deals with Mock objects
  - Creates the dependencies
  - Injects them in the tested class
  - Records Expectation objects.
  - Replays Expectations objects.
  - Executes the tested method.
  - Verifies any issue.

```
Method TestSumOperation()
{
  set mathService = ..CreateMock()
  set testedClass = ##class(MockDemo.CTestedClass).%New(mathService)

  do ..Expect(mathService.Add(1, 2)).AndReturn(3)
  do ..ReplayAllMocks()

  do $$$AssertEquals(testedClass.SumOperation(1, 2), 3)

  do ..VerifyAllMocks()
}
```

CTestedClass.cls | CTestTestedClass.cls

```
Class MockDemo.UnitTests.CTestTestedClass Extends Tests.Fw.CUnitTestBase
{

Property mathService As MockDemo.MathService [ Private ];

Property testedClass As MockDemo.CTestedClass [ Private ];

// --- Run test ---

ClassMethod RunTests()
{
    do ##super()
}

Method OnBeforeOneTest(testname As %String) As %Status
{
    set ..mathService = ..CreateMock()
    set ..testedClass = ##class(MockDemo.CTestedClass).%New(..mathService)
    quit $$$OK
}

// --- Tests for SumOperation ---

Method TestSumOperation()
```

```
Method TestSumOperation()
{
    do ..Expect(..mathService.Add(1, 2)).AndReturn(3)
    do ..ReplayAllMocks()

    do $$$AssertEquals(..testedClass.SumOperation(1, 2), 3)

    do ..VerifyAllMocks()
}
```

```
Method TestSumOperation()
{
    do $$$AssertEquals(..testedClass.SumOperation(1, 2), 3)
}
```

A MORE USEFUL EXAMPLE

```
Class MockDemo.CTestedClass Extends %RegisteredObject
{

Property mathService As MathService [Private];

Method %OnNew(mathService As MathService = {##class(MathService).%New()}) As %Status [ Private, Server
{
   set ..mathService = mathService
   quit $$$OK
}

Method ComplexOperation(A As %Integer, B As %Integer, ByRef status As %Status = {$$$OK}) As %Integer
{
   if (..mathService.GreaterThan(A, B))
   {
      quit ..mathService.Divide(A, B, .status)
   }
   else
   {
      quit ..mathService.Multiply(A, B, .status)
   }
}
```

```
/// TODO : Implement division some day
}
```

```
/// The mathService.Divide() method isn't even implemented. Yet this test is able
/// to verify that the status can be properly returned when passed by reference.
/// The result of 6/2 doesn't even count. The mock is told to return "50".
Method TestComplexOperationDivide()
{
   #Dim expectedStatus As %Status = $$$OK
   do ..Expect(..mathService.GreaterThan(6, 2)).AndReturn(1)
   do ..Expect(..mathService.Divide(6, 2, ..ByRefParam($$$OK, expectedStatus))).AndReturn(50)
   do ..ReplayAllMocks()

   #Dim status As %Status = $$$OK
   do $$$AssertEquals(..testedClass.ComplexOperation(6, 2, .status), 50)
   do $$$AssertStatusOK(status)

   do ..VerifyAllMocks()
}

Method TestComplexOperationMultiply()
{
   #Dim expectedStatus As %Status = $$$OK
   do ..Expect(..mathService.GreaterThan(6, 2)).AndReturn(0) // <-- Does not even have to be real.
   do ..Expect(..mathService.Multiply(6, 2, ..ByRefParam($$$OK, expectedStatus))).AndReturn(-25)
   do ..ReplayAllMocks()

   #Dim status As %Status = $$$OK
   do $$$AssertEquals(..testedClass.ComplexOperation(6, 2, .status), -25)
   do $$$AssertStatusOK(status)

   do ..VerifyAllMocks()
}
```

LET'S RECAP

- Replace a dependency by a controlled object
- This isolates the tested class
- Any scenario can be simulated

BONUS !

AN ENSEMBLE EXAMPLE

USING MOCK TO TEST OUR ENSEMBLE CODE

- Used to test
  - Services
    - Ens.BusinessService
    - Inbound Adaptor
  - Processes
    - Ens.BusinessProcess
  - Operations
    - Ens.BusinessOperation
    - Outbound Adaptor

AN ADAPTOR IS JUST A DEPENDENCY…

*CTestedPassthroughOperation.cls | CTestTestedPassthroughOperation.cls

```
Class EnsExt.File.CTestedPassthroughOperation Extends EnsLib.File.PassthroughOperation
{

/// initialize Business Host object
Method %OnNew(pConfigName As %String, adapterOverride As Ens.Adapter = {$$$NULLOREF}) As %Status
{
    do ##super(pConfigName)

    // Used for unit testing injection only.
    set:($IsObject(adapterOverride)) ..Adapter = adapterOverride

    quit $$$OK
}

}
```


...THAT CAN BE MOCKED AND INJECTED...

*CTestedPassthroughOperation.cls *CTestTestedPassthroughOperation.cls

```
Class Tests.Unit.EnsExt.File.CTestTestedPassthroughOperation Extends Tests.Fw.CUnitTestBase
{

Property operation As EnsLib.File.PassthroughOperation;

Property Adapter As EnsLib.File.OutboundAdapter;

ClassMethod RunTests()
{
    do ##super()
}

Method OnBeforeOneTest(testname As %String) As %Status
{
    set ..Adapter = ..CreateMock()
    set ..operation = ##class(chs.Fw.EnsExt.File.CTestedPassthroughOperation).%New("UnitTest", ..Adapter)
    set ..operation.Filename = "%f_%Q"

    quit $$$OK
}

// -- OnMessage tests --

Method TestOnMessage()
```

...AND THEN CONTROLLED !

```
Method TestOnMessageFailure()
{
    set stream = ##class(%Stream.GlobalCharacter).%New()
    do stream.Write("Test")
    set ensStream = ##class(Ens.StreamContainer).%New(stream)
    set ensStream.OriginalFilename = "C:/Temp/Blah.txt"

    do ..Expect(..Adapter.FilePath).AndReturn("C:/Temp/")
    do ..Expect(..Adapter.CreateFilename("Blah.txt", "%f_%Q")).AndReturn("Blah.txt_20170901...txt")
    do ..Expect(..Adapter.PutStream("Blah.txt_20170901...txt", stream)).AndReturn($$$ERROR($$$EnsErrGeneral, "PutStream()
    do ..ReplayAllMocks()

    do $$$AssertStatusNotOK(..operation.OnMessage(ensStream, .out))

    do ..VerifyAllMocks()
}
```


A PROCESS ALSO HAS DEPENDENCIES...

CTestedProcess.cls

```
Include chs.Fw.Defines

Class EnsExt.Lib.CTestedProcess Extends Ens.BusinessProcess [ ClassType = persistent ]
{

/// Configuration item to which to send file stream messages
Property TargetConfigName As %String(MAXLEN = 1000);

Property ensProcess As Ens.BusinessProcess [ Private ];

/// initialize Business Host object
Method %OnNew(pConfigName As %String, ensProcess As Ens.BusinessProcess = {$This}) As %Status
{
    set ..ensProcess = ensProcess

    quit ##super(pConfigName)
}

Method OnRequest(pRequest As Ens.Request, Output pResponse As Ens.Response) As %Status
{
    quit ..ensProcess.SendRequestAsync(..TargetConfigName, pRequest)
}

/// Handle a 'Response'
```

...THAT CAN BE MOCKED AND INJECTED

CTestedProcess.cls | CTestTestedProcess.cls

```
Class Tests.Unit.EnsExt.Lib.CTestTestedProcess Extends Tests.Fw.CUnitTestBase
{

Property testedProcess As EnsExt.Lib.CTestedProcess;

Property ensProcess As Ens.BusinessProcess;

ClassMethod RunTests()
{
    do ##super()
}

Method OnBeforeOneTest(testname As %String) As %Status
{
    set ..ensProcess = ..CreateMock()
    set ..testedProcess = ##class(EnsExt.Lib.CTestedProcess).%New("UnitTest", ..ensProcess)
    set ..testedProcess.TargetConfigName = "SomeTarget"

    quit $$$OK
}

// -- OnRequest tests --
```

CODE GIVEN TO COMMUNITY

- Reuses the mechanism of %UnitTest
  - Asserts
  - Test execution
- Built on top of the %UnitTests additions provided @ Summit 2015
- More examples in the library

ANY QUESTIONS ?

```
do ..Expect(presenter.SaysThankYou("en-US"))
do ..Expect(presenter.StopsTalking($NOW()))
do ..Expect(presenter.AnswersToQuestion(listOfQuestions))
do ..Expect(audience.Stands())
do ..Expect(audience.Claps()).Times(20)
do ..ReplayAllMocks()

do $$$AssertStatusOK(presentation.LastSlide())

do ..VerifyAllMocks()
```

Centre intégré universitaire de santé et de services sociaux de l'Estrie – Centre hospitalier universitaire de Sherbrooke

Québec

Please complete the survey in the event app.

Session recording and slides will be available at:
Learning.InterSystems.com
Search for "Global Summit 2017"

Visit the Tech Exchange to learn more!

The power behind what matters.

InterSystems®
Health | Business | Government

Thank you.