Xerris Bootcamp Series

Test Automation with Mocks

xerris

ACCOLITE DIGITAL
Transforming The Future, Now

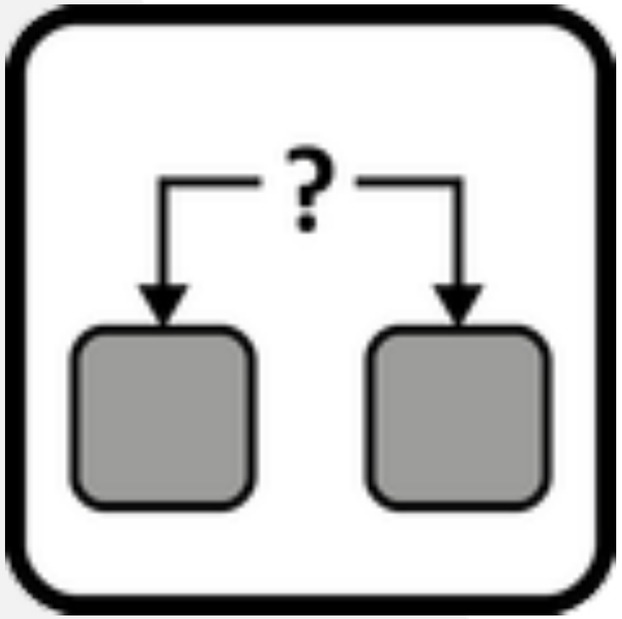
Test Automation with Mocks

What is Mocking?

- To “mock” is to make a replica of something.
- In a unit test, we need to test the methods of one class in isolation.
- What if that class’ methods depend on another object?
- Mocks allow you to get control over these dependencies to control the scenario you are testing.

Prerequisites for Mocking

- To support Mocking, classes need to be loosely coupled
- Classes cannot instantiate their dependencies within their constructor
- Usually done via Dependency Injection (Service Collection).
- Usually done by defining an interface for the dependency.



Test Automation with Mocks

What is Mocking?

- We need a way of ‘mocking’ the class dependencies in order to control their behaviour and outputs to test how this class responds to those outputs.

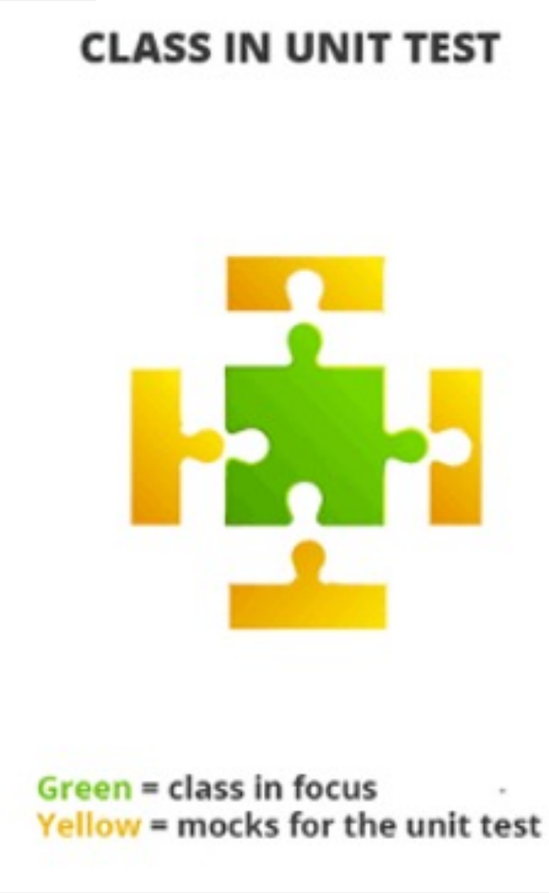
Moq

- Moq is a popular open-source mocking framework for .NET/C#.
- Moq provides many options for mocking dependencies.

The Mocking Workflow

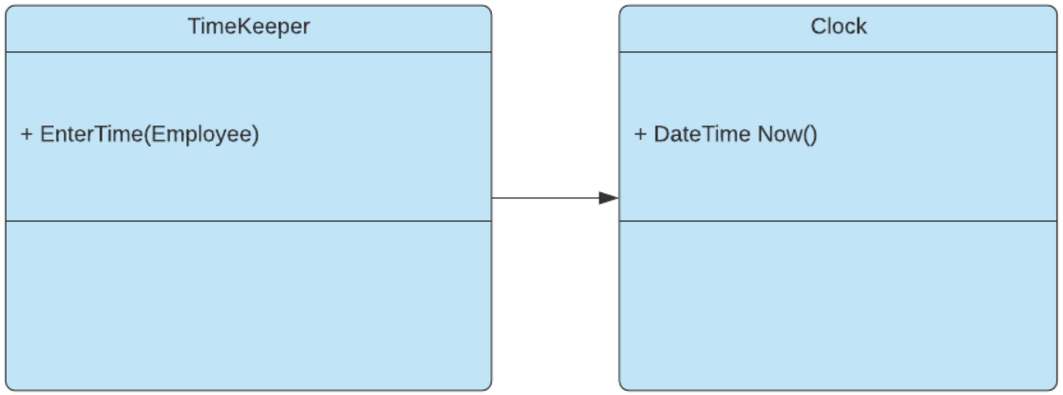
Here is a typical workflow when using most mocking libraries.

- Create/install the Mock
  create the mock and ‘install’ it in your class-under-test, usually through its constructor. (constructor injection)
- Configure the Mock
  Set up the mock with the calls and return values it should expect to receive from your class-under-test.
- Execute the Test
  Invoke the method-under-test and capture any outputs.
- Verify the calls to the Mock Object
  Ensure the calls to the mock are what you expected.

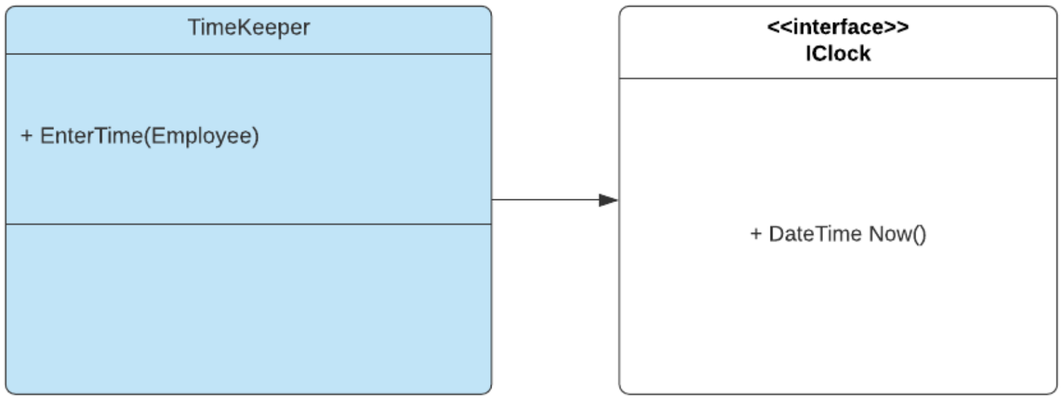


Mocking with Moq - Dependencies

How do we isolate the Timekeeper from the Clock class?



We use interfaces to represent these dependencies and allow for loose coupling



Mocking with Moq – Basic Mock Test

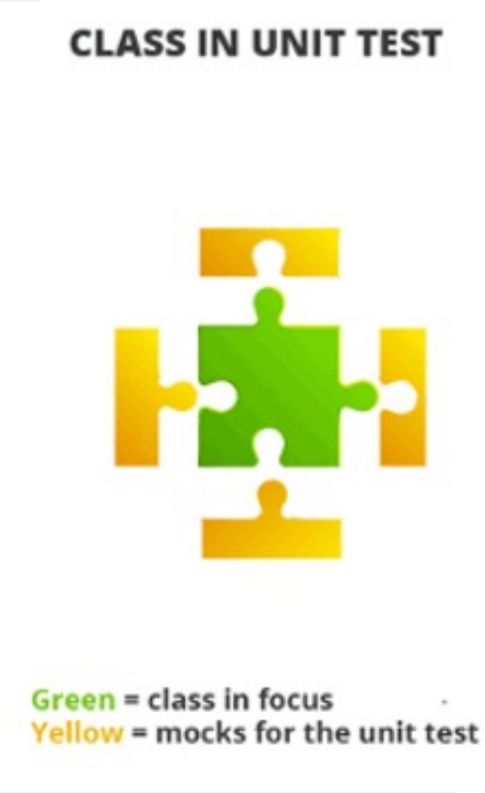
Simple set up for mocking the clock class.

```
public class TimeKeeperTest
{
    private MockRepository mocks;
    private Mock<IAlarmClock> mockClock;
    private TimeKeeper timeKeeper;

    public TimeKeeperTest()
    {
        mocks = new MockRepository(MockBehavior.Strict);

        mockClock = mocks.Create<IAlarmClock>();
        timeKeeper = new TimeKeeper(mockClock.Object);

        mockClock.SetupGet(x => x.Now).Returns(DateTime.Now);
    }
}
```



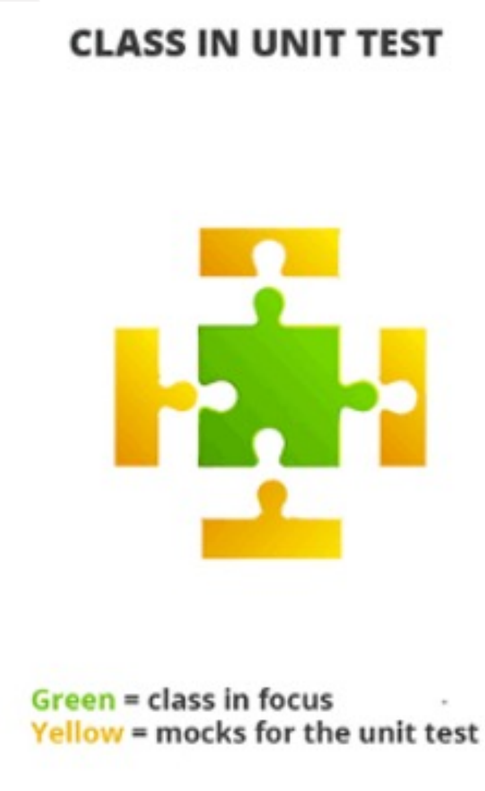
Mocking with Moq – Basic Mock Test

Create & Install the Mock.

```
    private MockRepository mocks;
    private Mock<IAlarmClock> mockClock;
    private TimeKeeper timeKeeper;

    public TimeKeeperTest()
    {
        mocks = new MockRepository(MockBehavior.Strict);

        mockClock = mocks.Create<IAlarmClock>();
        timeKeeper = new TimeKeeper(mockClock.Object);
    }
}
```



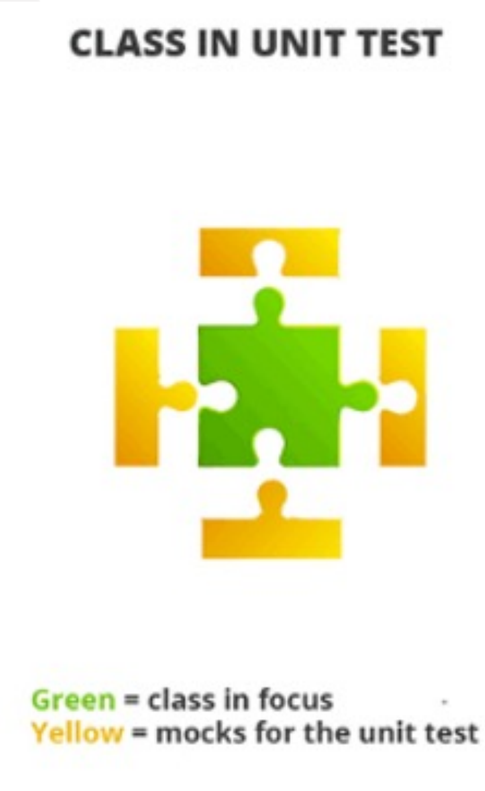
Mocking with Moq – Basic Mock Test

Invoke the test method

```
{
    ...
    public TimeKeeperTest()
    {
        timeKeeper = new TimeKeeper(mockClock.Object);
        timeKeeper.WhatTimeIsIt();
    }
}
```

Verify the Mock

```
public TimeKeeperTest()
{
        mockClock.VerifyGet(x => x.Now);
}
```



Mocking with Moq – Verification

Moq provides may ways to verify calls to the mock.

```
mock.Verify(foo => foo.DoSomething("ping"));

// Verify with custom error message for failure
mock.Verify(foo => foo.DoSomething("ping"), "not what I expected");

// Method should never be called
mock.Verify(foo => foo.DoSomething("ping"), Times.Never());

// Called at least once
mock.Verify(foo => foo.DoSomething("ping"), Times.AtLeastOnce());

// Verify getter invocation, regardless of value.
mock.VerifyGet(foo => foo.Name);

// Verify setter with an argument matcher
mock.VerifySet(foo => foo.Value = It.IsInRange(1, 5, Range.Inclusive));

// Verify that no other invocations were made
mock.VerifyNoOtherCalls();

// Verifies ALL expected calls to this mock.
// Reports calls that were made and not expected
mock.VerifyAll();
```

Mocking with Moq – Workshop

Music Handler Lambda

- Write a unit test for the MusicHandler
- Implement the mock needed to invoke the MusicService

Steps

- Setup/Install the Mock
- Program the mock for each test
- Invoke the method-under-test in each test
- Verify the calls to the mock in each test