Mocking in testing

What is mocking?

- To ”mock” it 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 dependencies in order to control the scenario you are testing.

xerris

Mocking in testing

What is mocking?

- We need a way of ‘mocking’ the class’ dependencies in order to get control over them in a unit test.

Moq

- Is a popular open-source mocking framework for C#
- Moq provides many options for mocking dependencies.
- We will explore many here

xerris

Mocking Workflow

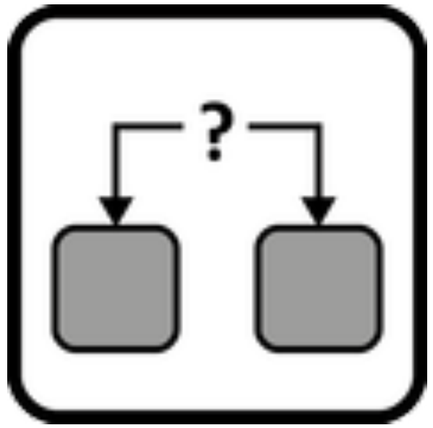
What steps do you take when using Mocks?

- Create/Install the Mock
  - Create the mock and ’install’ it in your class-under-test
- Configure the Mock
  - Setup the mock with the calls and return values it should expect to receive from your class-under-test
- Execute the test
- Verify the calls to the Mock

xerris

Allowing Mocking

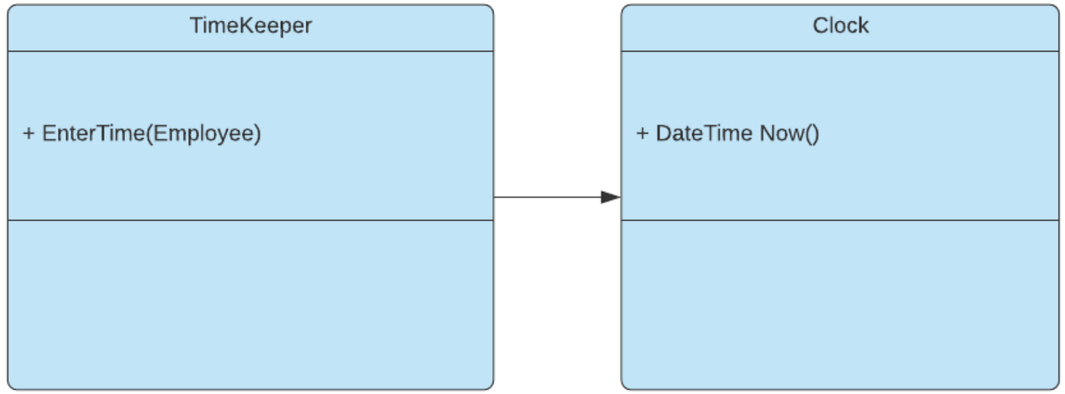
- To support Mocking classes need to be **loosely coupled**
- Classes cannot instantiate their dependencies
- Usually done via Dependency Injection (ServiceCollection)
- Usually done by defining an Interface for the dependency



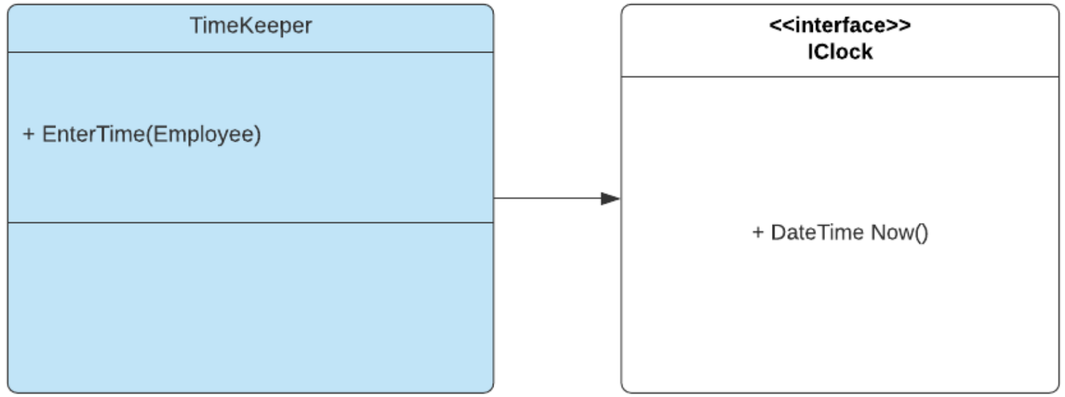
xerris

Mocking in testing

- How do we isolate the TimeKeeper from the Clock dependency?



- We use Interfaces to represent these dependencies but allow loose coupling



xerris

Basic Mock Test

- Basic Setup for mocking using Moq

```
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);
    }
}
```

xerris

Basic Mock Test

Mock Setup

- Program it with the expectations

```
var clock = new Mock<IAlarmClock>();
clock.SetupGet(p => p.Now).Returns(DateTime.Today);
clock.Setup(p => p.SetAlarm(new DateTime("…")));
```

Verify The Mock

- You can ask the mock if the expected calls were made

```
mockClock.VerifyGet(x => x.Now);
```

xerris

Argument Matching

```
// any value
mock.Setup(foo => foo.DoSomething(It.IsAny<string>()))
        .Returns(true);

// any value passed in a `ref` parameter (requires Moq 4.8 or later):
mock.Setup(foo => foo.Submit(ref It.Ref<Bar>.IsAny)).Returns(true);

// matching Func<int>, lazy evaluated
mock.Setup(foo => foo.Add(It.Is<int>(i => i % 2 == 0))).Returns(true);

// matching ranges
mock.Setup(foo => foo.Add(
      It.IsInRange<int>(0, 10, Range.Inclusive))).Returns(true);

// matching regex
mock.Setup(x => x.DoSomethingStringy(
      It.IsRegex("[a-d]+", RegexOptions.IgnoreCase))).Returns("foo");
```

xerris

Verification

```
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();
```

xerris

Mocking Workshop

Music Handler?

- Write the tests for the Music Handler
- Implement the mocks needed to test each method

Steps

- Setup/Install the Mock
- Configure the Mock for each test
- Invoke the class-under-test
- Verify the Mock

xerris