Creating a Mock Instance

- Classes to be mocked must be an interface or have methods that can be overridden
- `var mockInstance = new Mock<ClassToMock>();`

Accessing the Actual Mocked Object

- Use **`.Object`** on the mock object instance
- `mockInstance.Object`

Making Simple Checks

- Check that a method was called
  - Arrange
    - `mockInstance.SetUp(x => x.MethodToCall());`
  - Assert that `MethodToCall()` was called
    - `mockInstance.Verify();`
  - Or without the setup
    - Assert that `MethodToCall()` was called
      - `mockInstance.Verify(x => x.MethodToCall());`
- Check how many times a method was called
  - Assert that `MethodToCall()` was called exactly 1 time
    - `mockInstance.Verify(x => x.MethodToCall(), Times.Exactly(1));`

Passing Arbitrary Arguments

- Use **`It.IsAny<Data Type>()`**
- `mockInstance.SetUp(x => x.MethodToCall(It.IsAny<int>()));`

Returning Values

- If the return was not explicitly mocked then the C# default value for that type will be returned.
- Simple returns
  - `mockInstance.SetUp(x => x.MethodToCall()).Returns(() => null); // with lambda`
  - `mockInstance.SetUp(x => x.MethodToCall()).Returns(null);`
- Out parameters
  - Create the object to be returned
    - **`var someObject = new SomeClass();`**
  - Pass object into the method
    - Assume `MethodToCall()` has this signature:
      - `void MethodToCall(string str, out SomeClass obj);`
    - `mockInstance`
      `.SetUp(x => x.MethodToCall(It.IsAny<string>(), out someObject));`
- Return a different value each time the method is called
  - Assume that each time `MethodToCall()` is called in the test, it returns a different integer.
  - ```
    var i = 0;
    mockInstance.SetUp(x => x.MethodToCall())
          .Returns(() => i)
          .CallBack(() => i++);
    ```
    ```

Arguments

- Ensuring that correct values are being passed to dependencies
  - ```
    mockInstance
          .SetUp(x => x.MethodToCall(It.IsAny<string>(), It.IsAny<string>()));
    ```
  - ```
    mockInstance
          .Verify(
                x => x.MethodToCall(
                      It.Is<string>(s=>s.Equals(someString)),
                      It.Is<string>(s=>s.Equals(someOtherString))));
    ```
- Control the flow of the code
  - ```
    mockInstance
          .SetUp(x => x.MethodToCall(
                It.Is<SomeClass>(y => y.State == SomeDesiredState)))
          .Returns(DesiredState);
    ```

Exceptions

- Setting up SUT exception throwing
  - ```
    mockInstance
          .SetUp(x => x.MethodToCall())
          .Throws<SomeException>();
    ```

Properties

- Verify setter was called
  - ```
    mockInstance
          .VerifySet(x => x.SomeProperty = It.IsAny<string>());
    ```
- Return values from getters and verifying that it was called
  - ```
    mockInstance
          .SetUp(x => x.SomeProperty).Returns(someValue);
    ```
  - ```
    mockInstance
          .VerifyGet(x => x.SomeProperty);
    ```
- Auto-mocking hierarchies e.g. `obj.Prop1.Prop2.Prop3` (The properties must be overrideable.)
  - ```
    mockInstance
          .SetUp(x => x.Prop1.Prop2.Prop3).Returns(someValue);
    ```
  - mockInstance
    .VerifyGet(x => x.Prop1.Prop2.Prop3);

Stubbing Properties

- Pre-set values for properties on mock objects
  - ```
    mockInstance
          .SetupProperty(x => x.SomeProperty, someValue);
    ```
- Changing those values
  - `mockInstance.Object.SomeProperty = someOtherValue;`
- SetupAllProperties
  - `mockInstance.SetupAllProperties(); // All properties are now stubbed`

Events

- Raise on the mock
    ```

    - mockInstance
      .Raise(x => x.SomeEvent += null, eventArgs);
- Non-standard event signatures
    - mockInstance
      .Raise(x => x.SomeEvent += null, **arg1, arg2**);