# Unit Testing in C# with nUnit – Level 3

In this set of work we extend beyond Level 1 and Level 3 by using Mocks. Please complete Level 1 and 2 prior to working on this Level.

At the time of writing this there were three or four fairly popular libraries in the market to choose from, I chose to go with Moq out of its simplicity for the API and google trends. Right click the BankTest project, and Manage nuget packages. Search for and add in the “Moq” library.

For simplicity let’s give the library the ability to write the Bank account out to a file. Since we are introducing Mocks I will delay the unit testing until we can see the structure that is setup.

Create a BankAccountWriter class which will allow the user to take the BankAccount and write it to disk:

BankAccountWriter.cs

using System;

namespace Bank

{

public class BankAccountWriter

{

private readonly IFileWriter fileWriter;

public BankAccountWriter(IFileWriter fileWriter)

{

this.fileWriter = fileWriter;

}

public void WriteAccount(BankAccount account)

{

fileWriter.Write(account.Name, String.Format("{0}|{1}", account.Name, account.Balance));

}

public BankAccount ReadAccount(string name)

{

BankAccount account = null;

var contents = fileWriter.Read(name);

var both = contents.Split('|');

if (both.Length == 2)

{

double balance = 0;

if (double.TryParse(both[1], out balance))

{

account = new BankAccount(both[0], balance);

}

}

return account;

}

}

}

As you can see, this class takes an IFileWriter interface in its constructor and then when “WriteAccount” is called, it formats the string and passes it to the fileWriter field. The IFileWriter interface is very simple and looks like this:

IFileWriter.cs

namespace Bank

{

public interface IFileWriter

{

void Write(string file, string contents);

string Read(string file);

}

}

By using this interface we have isolated the actual writing of the file to the disk from the formatting of the data that is going to be written to the disk. Obviously we will also need an implantation of our interface which will actually write to disk, here it is:

namespace Bank

{

public class FileWriter : IFileWriter

{

public void Write(string file, string contents)

{

System.IO.File.WriteAllText(file, contents);

}

public string Read(string file)

{

return System.IO.File.ReadAllText(file);

}

}

}

And a unit test around this:

[Test]

public void DebitAccount\_WriteToFile\_ThenRead()

{

// arrange

double beginningBalance = 11.99;

double debitAmount = 4.55;

double expected = 7.44;

BankAccount account = new BankAccount("Mr. Bryan Walton", beginningBalance);

account.Debit(debitAmount);

// act

IFileWriter writer = new FileWriter();

BankAccountWriter baw = new BankAccountWriter(writer);

baw.WriteAccount(account);

// assert

var readAccount = baw.ReadAccount(account.Name);

Assert.AreEqual(readAccount.Balance, account.Balance);

Assert.AreEqual(readAccount.Name, account.Name);

}

Keep in mind that during our unit testing we typically do not want to write anything to disk so in order to achieve this we will mock out the specific implementation for the IFileWriter interface as such.

[Test]

public void DebitAccount\_WriteToFile\_ThenRead\_Mocked()

{

// arrange

double beginningBalance = 11.99;

double debitAmount = 4.55;

double expected = 7.44;

BankAccount account = new BankAccount("Mr. Bryan Walton", beginningBalance);

account.Debit(debitAmount);

// act

var writer = new Moq.Mock<IFileWriter>();

writer.Setup(w => w.Write(Moq.It.IsAny<string>(), Moq.It.IsAny<string>()));

writer.Setup(w => w.Read(Moq.It.IsAny<string>())).Returns(String.Format("{0}|{1}", account.Name, account.Balance));

BankAccountWriter baw = new BankAccountWriter(writer.Object);

baw.WriteAccount(account);

// assert

var readAccount = baw.ReadAccount(account.Name);

Assert.AreEqual(readAccount.Balance, account.Balance);

Assert.AreEqual(readAccount.Name, account.Name);

}

The core of what we are mocking up is the actual implantation of IFileWriter. So you see we new up a new Mock with the interface IFileWriter:

var writer = new Moq.Mock<IFileWriter>();

And then on that Mock, we setup our Write and Read methods:

writer.Setup(w => w.Write(Moq.It.IsAny<string>(), Moq.It.IsAny<string>()));

We allow for any inputs for the Write method.

writer.Setup(w => w.Read(Moq.It.IsAny<string>())).Returns(String.Format("{0}|{1}", account.Name, account.Balance));

The read method will return a properly formatted string, with the BankAccountWriter class will know how to parse and return our Account object back to the caller.