# *Programming IV (420-B42-HR)*

# *Lab 3 – Unit Testing*

Date assigned: Monday, February 6, 2017

Date due: **Monday, February 6, 2017, 11:00 a.m.**

**Learning Objectives**

Upon successful completion of this lab exercise, the student will be able to:

* Create a Unit Test project to test class libraries
* Develop unit tests to debug class libraries

Lab Set-Up

1. Make sure you name the controls properly. See the Moodle page for standards on how the controls are to be named.
2. Create a folder called yourusername\_B42L03. All three test projects solutions for this project should go in this folder.
3. Download the TestLibraries zip file and unzip it as a subfolder of the above folder (that is yourusername\_B42L03/TestLibraries). There should be three solutions in the folder.

To Do

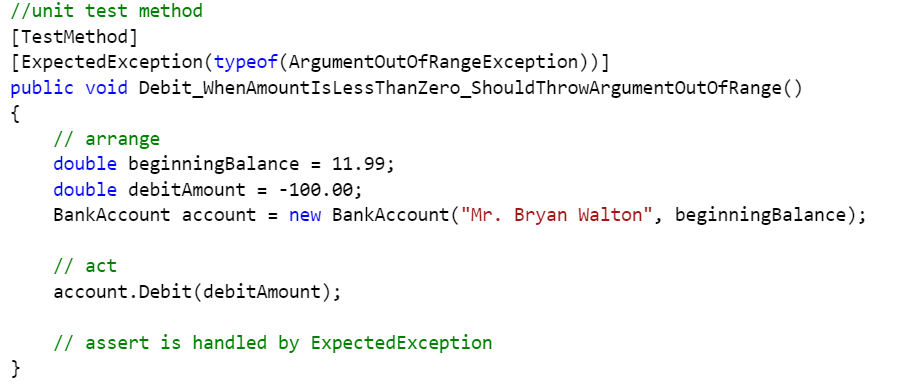
**Part A – Keep It Going**

1. We’re going to start with the project we worked on in class. Use Visual Studio and to create a new solution with a Test => Unit Test Project (File => New => Project). Choose Visual C# => Test => Unit Test Project. Name the project *yourinitialsPartAShapes* (and the solution).
2. You will need to add a reference to the class library that you just unzipped in the subfolder. This is the starting library from class (so before any of the changes were made).
3. Add new Test Classes to the test project we started in class (or create them from scratch). Test the calculation of Area, Perimeter and Volume. Make sure you test all the different shapes and all the different methods. Also, test the error that is caught with the triangle perimeter. There are at least 7 errors that can be found (including two in the first constructor). Note: you need to test by creating each of the shapes using both the complete constructor and the empty constructor and assigning values as required.
4. Document each error at the location of the error in the class library by commenting out the line of code and including what the error is. Then, fix the error. Re-run the tests until everything passes.

NOTE: There is a document in the files folder which describes what the methods in the class **should** return. You will need to use a calculator to determine the expected values and use that MANUALLY calculated value in your assertion.

NOTE 2: I KNOW the class is poorly defined and I would not create a class like this in real life. I used many inefficiencies and poor practices. Do NOT correct the layout of the class or rewrite the class. Just find and correct the errors as is.

**Part B – Bank Account**

1. Create a new unit test project called *yourinitialsB42L03B* and add a reference to the BankLibrary DLL from the BankLibrary solution folder you copied.
2. Open the class library and look at the code and determine what the test cases should be. There are four methods to test: Debit subtracts a passed amount from the account if funds are available; Credit adds a passed amount to the account if funds are provided. FreezeAccount freezes the account and UnfreezeAccount sets the account as not frozen.
3. There are at least 4 errors to find (NOT including the error that you cannot set the account name or starting balance if you use the empty constructor. DO NOT use the empty constructor). A couple are somewhat subtle, so look carefully.
4. Once again, document the errors in the library file, correct the error and re-run the tests until all the errors are corrected.
5. This library throws exceptions whenever there are errors. This is a common method of dealing with errors. To test that the proper error is thrown, you tell the method what error you expect to receive before the test as in the example below:  
     
     
   Notice that there is no Assert used. The ExpectedException directive acts as an Assert. To see it fail, change the exception in the library, recompile and run your test and it will fail.
6. The other option to dealing with exceptions being thrown is to deal with it directly in the TestMethod. To do this, place call to the library method that will throw the exception inside a try/catch and use a StringAssert in the catch to make sure that the proper exception was caught.
7. You must use at least one of each way to catch exceptions in testing the banking library.

**Part C – Moving Expenses**

1. The final class library to test is a library of moving expenses. The library instantiates an object of type moving. The constructor takes either no parameters, two parameters for distance and weight or all five possible parameters.
2. There is only one public method which calculates the total cost of moving. The public method calls 5 private methods and adds their returns together to get the total cost. There are comments before each private method which CORRECTLY indicate how each method calculates the amounts.
3. Look at the class library called MovingLibrary which you copied at the beginning of the lab. Create a unit test project called *yourinitialsB42L03C* and add a reference to the MovingLibrary DLL from the MovingLibrary solution folder.
4. Create test methods to test all the methods in the class. There are at least 6 errors to be found. Each of the errors is very common, although they may not all appear at once. Once again, document them where they are located in the class library and correct the errors. YOU WILL HAVE TO DO SOME MATH USING A CALCULATOR TO SEE IF YOUR AMOUNTS ARE CORRECT. I suggest working at this one step at a time.

**To submit**

When you have completed the lab exercise, create a single zip file called YourUserName\_B42\_L03.zip and copy the file to the Moodle page for the course.