**Testing and Debugging**

**Lab Practice**

Advanced Software Engineering

Thalita Vergilio

## Contents

[Contents 2](#_Toc23260755)

[What you need 2](#_Toc23260756)

[1. Using the Visual Studio Unit Testing Framework 3](#_Toc23260757)

[1.1. Clone the project from BitBucket 3](#_Toc23260758)

[2.2. Task 5](#_Toc23260759)

[2. Using the XUnit.net Unit Testing Framework 10](#_Toc23260760)

[2.1. Clone the project from BitBucket 10](#_Toc23260761)

[2.2. Task 12](#_Toc23260762)

[Optional: Install Dependencies 13](#_Toc23260763)

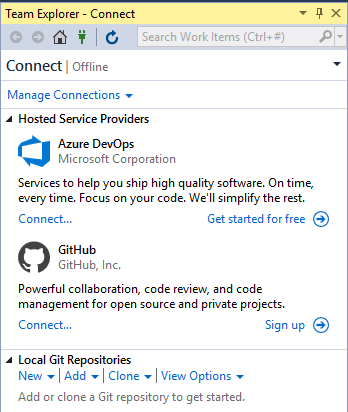
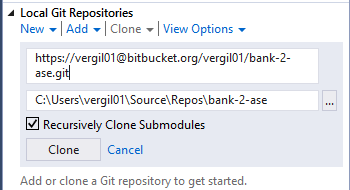
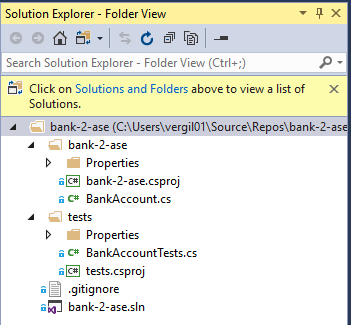
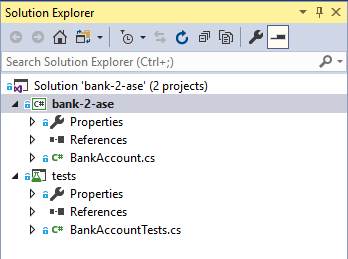
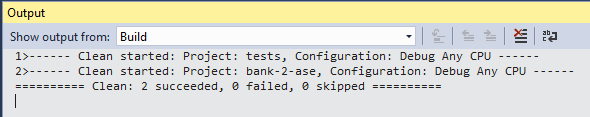
## What you need

* Visual Studio
* .NET Framework
* Git

## 1. Using the Visual Studio Unit Testing Framework

In this section, you will practice writing unit tests and implementation code using a test-driven approach to development. You will use the Visual Studio Unit Testing Framework.

### 1.1. Clone the project from BitBucket

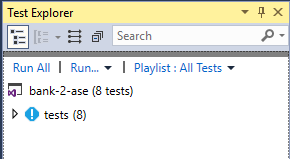
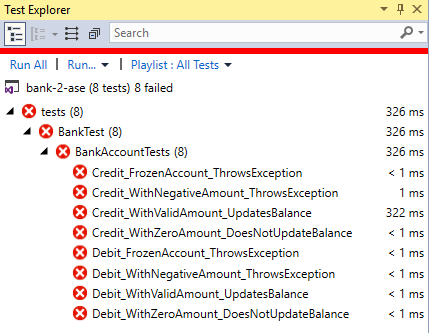
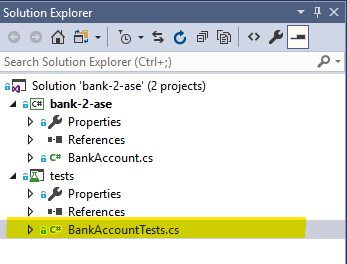
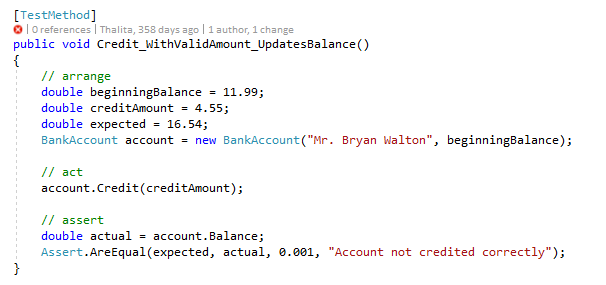
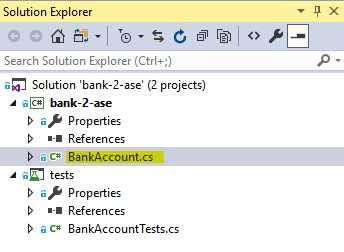
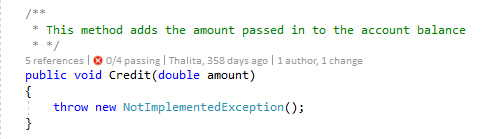
1. Open Visual Studio, navigate to File/Open/Open from Source Control.
2. The Team Explorer window will open.  
   
3. Under "Local Git Repositories", select "Clone".
4. Enter the following URL: https://vergil01@bitbucket.org/vergil01/bank-2-ase.git
5. Enter the destination folder (i.e. C:\Users{YOU}\Source\Repos\bank)
6. Leave the “Recursively Close Submodules” checkbox selected.  
   
7. Click on “Clone”.
8. Select View/Solution Explorer. Check that the project has been cloned correctly and you can see the files.  
   
9. Double-click on bank-2-ase.sln to open the solution.  
   
10. In Solution Explorer, right-click on the Solution (on top) and select “Clean Solution”.  
    
11. Right-click on the Solution again and select “Build Solution”. Verify that the build was successful.

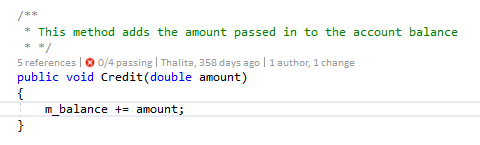
### 2.2. Task

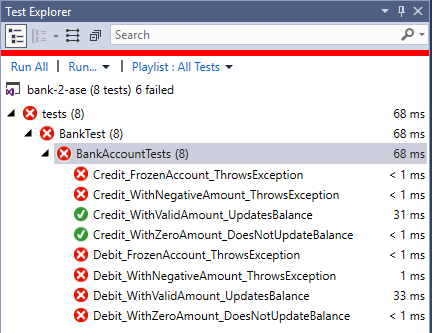
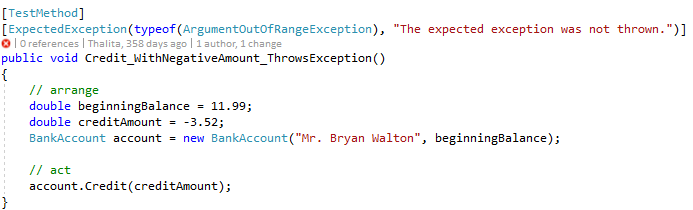
Your task is to implement two simple operations for the BankAccount class: credit() and debit(). You will use a test-driven approach:

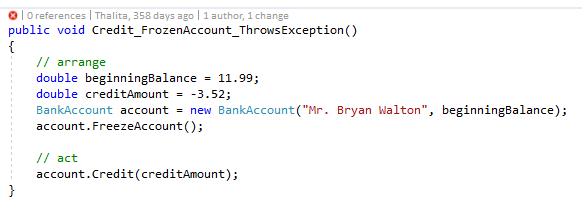
* Tests are written before the implementation.
* When first run, all tests are expected to fail.
* Implement one feature at a time, writing just enough code to pass the tests.
* Re-run the tests until they all pass.
* Refactor the code, making sure the tests still pass.

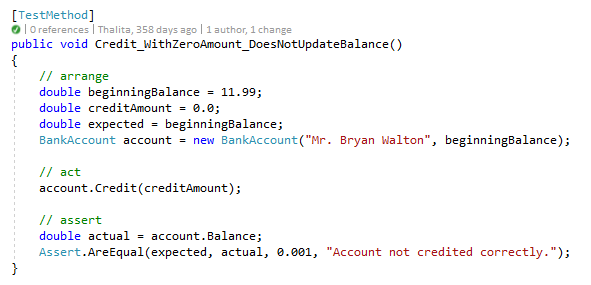
#### Steps

1. Navigate to Test/Windows/Test Explorer to open the Test Explorer.  
   
2. From the top of the Test Explorer, click on "Run All". You should see the results indicating that all your tests have failed. If your tests don’t run, skip to [Install Dependencies](https://docs.google.com/document/d/1McdRRgShFvekDawv1F5_y93O3YPvpPg8/edit#heading=h.3j2qqm3).  
   
3. Open the BankAccountTests.cs class by double-clicking on it.   
   
4. Look at the Credit\_WithValidAmount\_UpdatesBalance() method. See if you understand what the test is trying to do. Which method of which class is being tested?  
   
5. Open the BankAccount.cs class.  
   
6. Find the method being tested and note that it currently throws a NotImplementedException.   
   
7. Delete the exception and write a simple implementation - just enough to pass the test.



1. From your Test Explorer, click on "Run All" again. You will see that at least one of the tests is now green. If they are all still red, change the implementation again until the test passes.  
     
   
2. Repeat steps 3-4 for the Credit\_WithNegativeAmount\_ThrowsException() test. Hint: you will need to check for a condition in your implementation and throw an exception if that condition is true.  
   
3. Now repeat steps 3-4 for the Credit\_FrozenAccount\_ThrowsException() test.

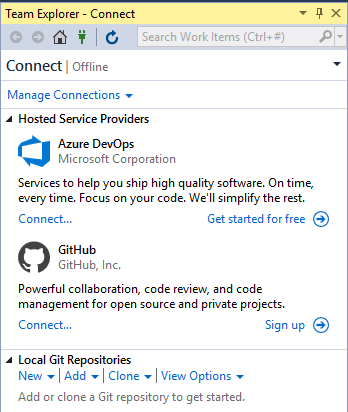


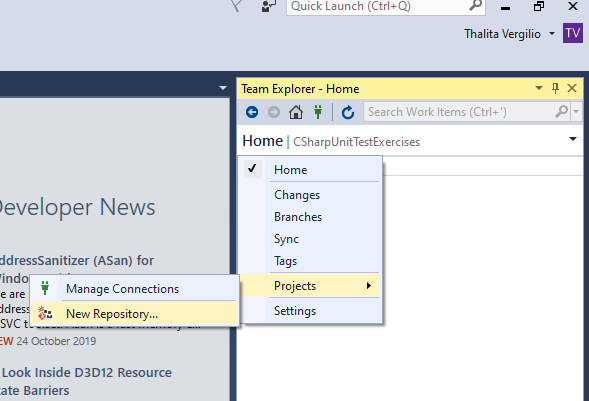
1. If the Credit\_WithZeroAmount\_DoesNotUpdateBalance() test is still red (it probably won't be, it depends on how you implemented step 3), repeat steps 3-4 for it to make sure it passes.  
   
2. Now it's your turn to write your own tests and then write the implementation code. Follow the examples given for the credit operation and implement the reverse: a debit operation. Make sure you use the test-driven approach outlined at the start of this document.  
   

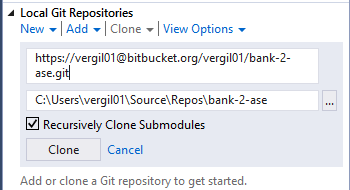
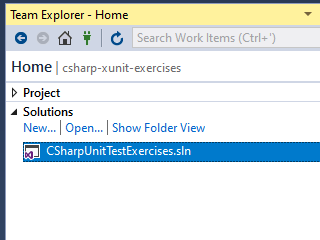
## 2. Using the XUnit.net Unit Testing Framework

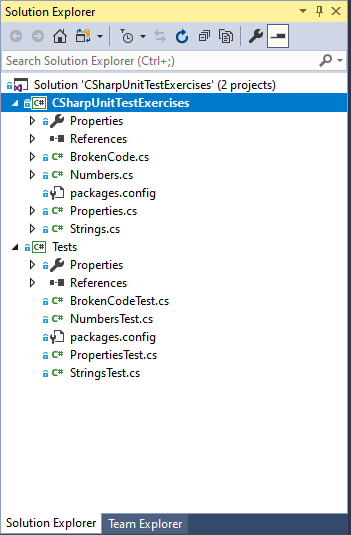
In this section, you will practice writing unit tests using a different framework. XUnit.net is an open-source testing framework related to JUnit, PHPUnit, PyUnit, and others. It is based on Facts (for testing invariable conditions) and Theories (for testing that assertions are true for specified arguments passed in).

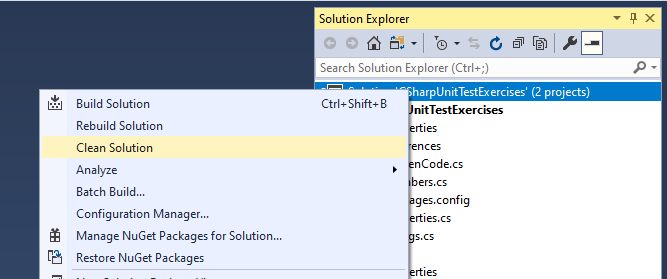
### 2.1. Clone the project from BitBucket

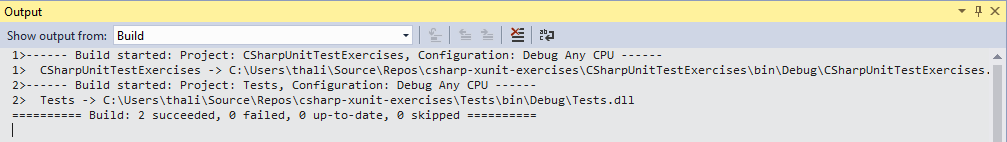
1. Open Visual Studio, navigate to File/Open/Open from Source Control.
2. The Team Explorer window will open.  
   

If you don’t see the window above, click on Home, then select Projects/New Repository.  


1. Under "Local Git Repositories", select "Clone".
2. Enter the following URL: https://vergil01@bitbucket.org/vergil01/csharp-xunit-exercises.git
3. Enter the destination folder (i.e. C:\Users{YOU}\Source\Repos\csharp-xunit-exercises)
4. Leave the “Recursively Close Submodules” checkbox selected.  
   
5. Click on “Clone”.
6. Double-click on the project cloned and double-click on the Solution to open it.  
   
7. Navigate to View/Solution Explorer and make sure you can see the files below.



1. In Solution Explorer, right-click on the Solution (on top) and select “Clean Solution”.  
   
2. Right-click on the Solution again and select “Build Solution”. Verify that the build was successful.



### 2.2. Task

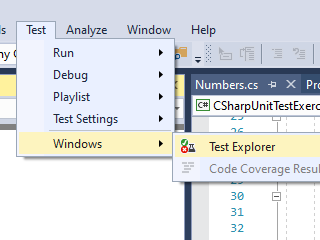
Your task is to get all the tests to pass. You will write (or correct) implementations for the methods in the exercise. There are four implementation classes and four test classes, as per the table below:

|  |  |
| --- | --- |
| Class | Test |
| BrokenCode | BrokenCodeTest |
| Numbers | NumbersTest |
| Strings | StringsTest |
| Properties | PropertiesTest |

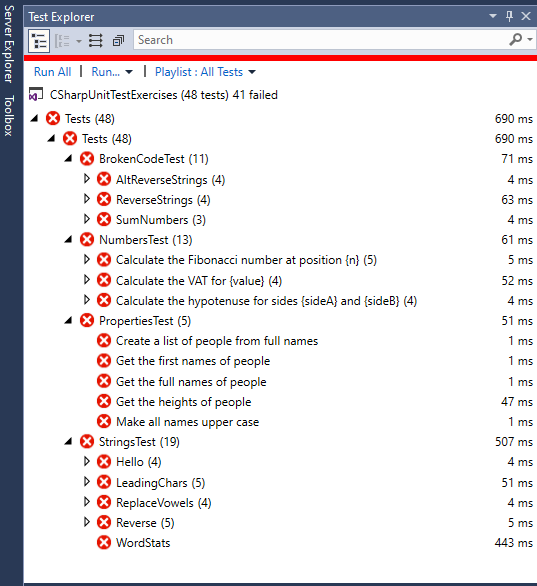
* You can complete the exercise in any order.
* Leave the ones marked “Advanced” to the end.
* Use the Debugger and breakpoints to understand what the code is doing at runtime.
* Implement one feature at a time, writing just enough code to pass the tests.
* Re-run the tests until they all pass.
* Refactor the code, making sure the tests still pass.

Steps

Navigate to Test/Windows/Test Explorer to open the Test Explorer.



Click on “Run All” and ensure that the tests complete and that they fail.



You can now start working on them. Have fun!

Optional: Install Dependencies

*Follow these steps if you’re experiencing problems running your tests.*

1. Check that you have all the dependencies the project needs. Right-click on the solution and select “Manage NuGet Packages for Solution”.
2. Ensure you are running the latest stable version for all dependencies. If not, install the missing ones.