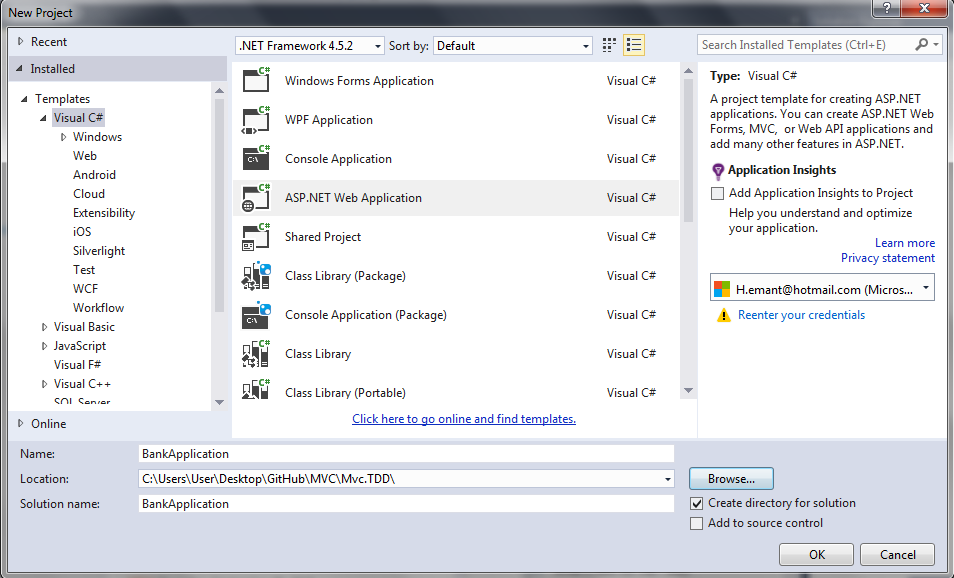
**TDD – ASP.NET MVC**

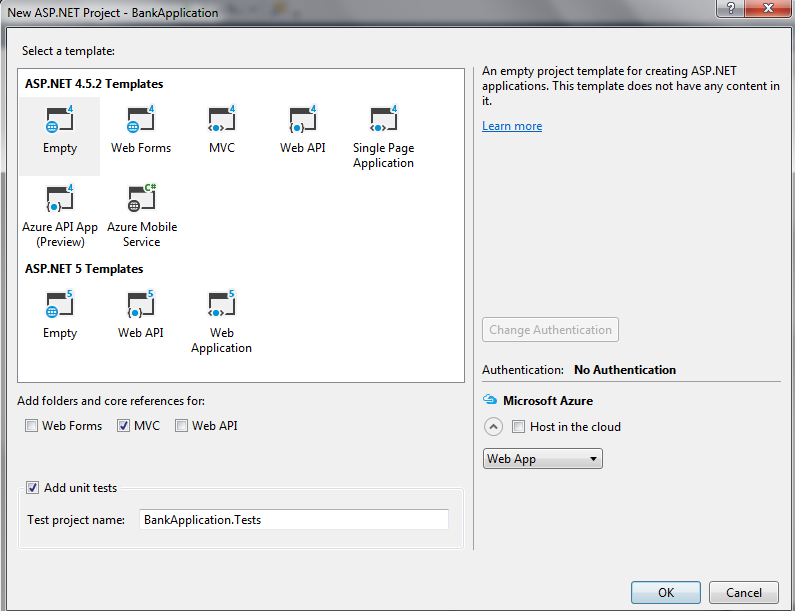
We have to design a loan application with following 3 requirements:

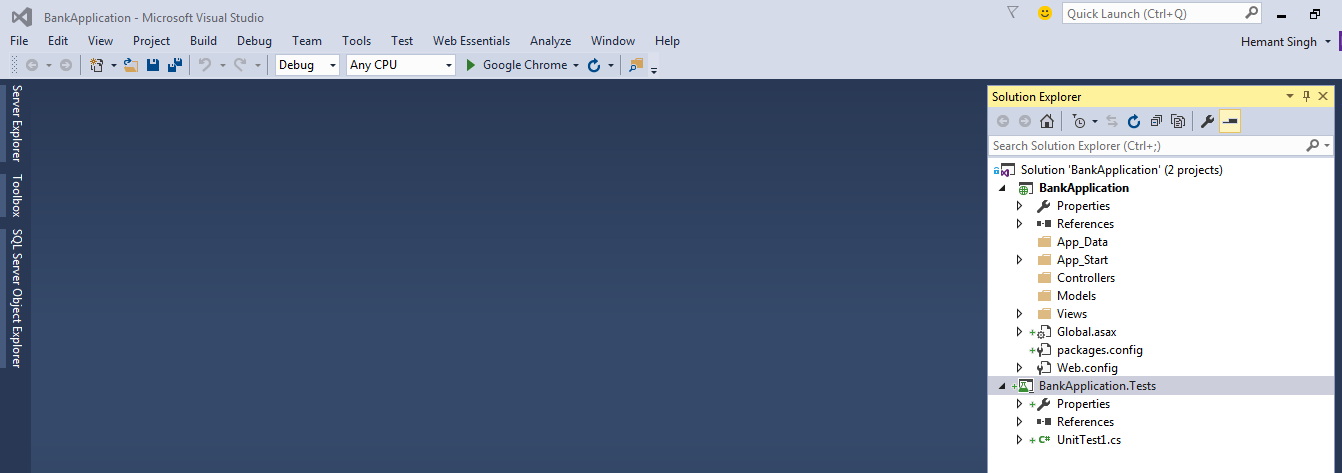
1. Verify customer’s loan eligibility
2. Verify customer’s credit history
3. Provide a UI to apply for loan
4. **Verify customer’s loan eligibility**

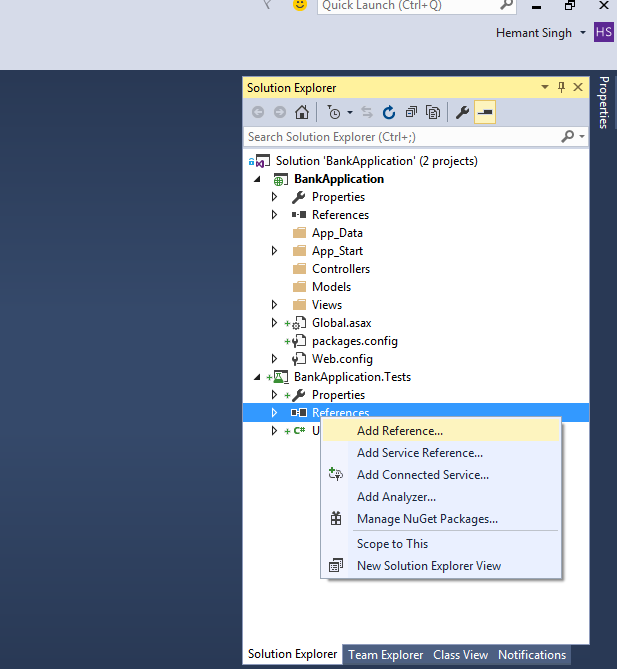
Following are the guidelines:

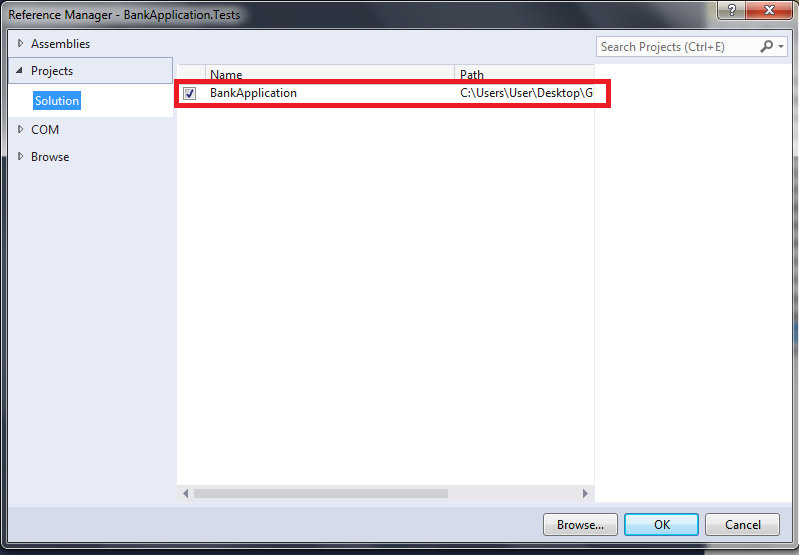
* The loan type should be either ‘Personal’ or ‘Car’
* The loan duration should be between 1 and 5 years
* The loan amount should be less than 40% of the annual income of the customer

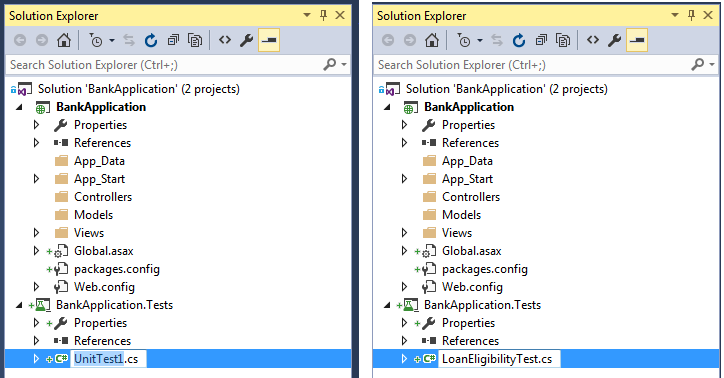


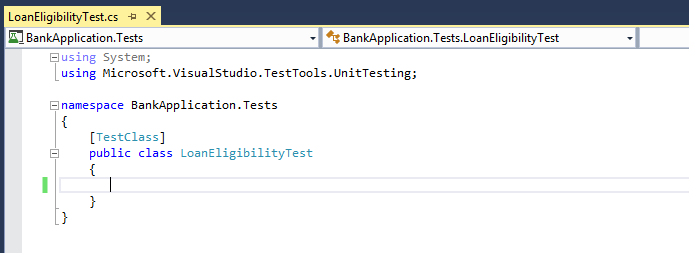












[TestClass]

public class LoanEligibilityTest

{

[TestMethod]

public void TestLoanTypePersonal()

{

LoanEligibility loanEligibility = new LoanEligibility();

//Arrange

string loanType = "Personal";

//Act

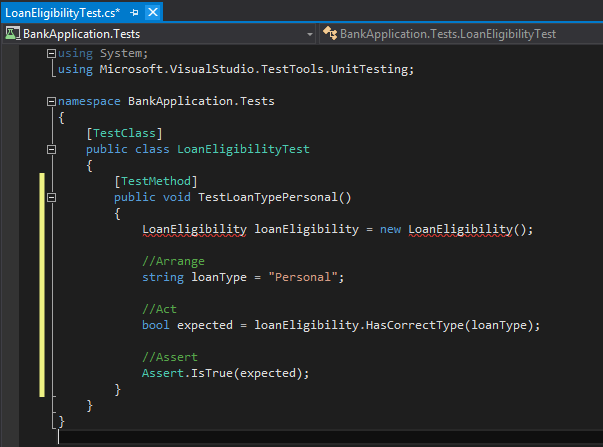
bool expected = loanEligibility.HasCorrectType(loanType);

//Assert

Assert.IsTrue(expected);

}

}



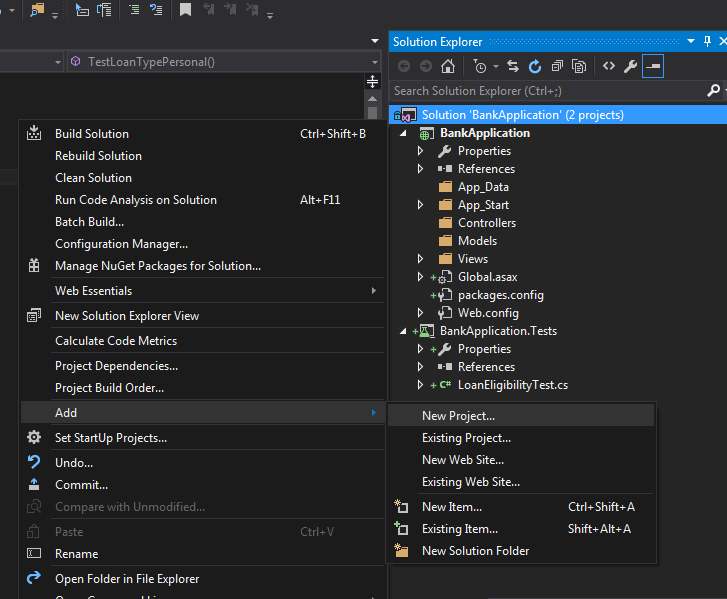
Here, LoanEligibility and its corresponding method HasCorrectType() are supposed to be methods that are yet to be defined in our main project.

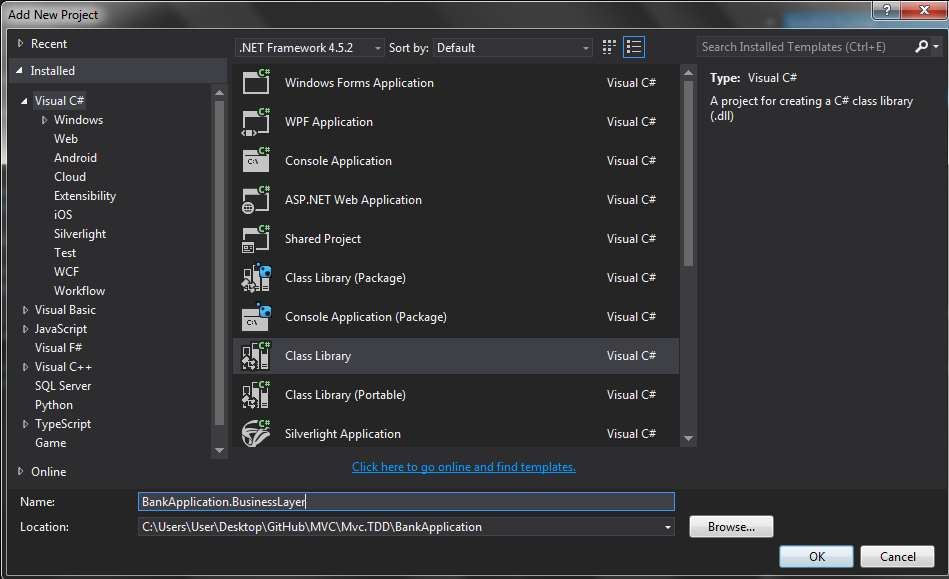
Obviously, right now, our test will give compile time error due to these undefined methods.

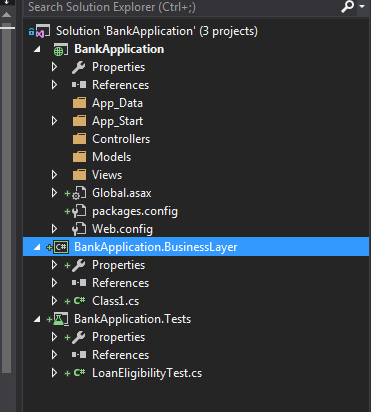
TDD states that as soon as your Unit Test fails, go to your main project and write the minimum source code that passes the test.

Let’s go to our main application and add the missing classes and methods.

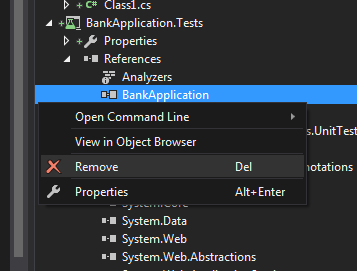
Let’s add a new project named BankingApplication.BusinessLayer

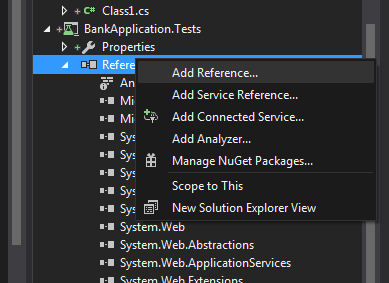


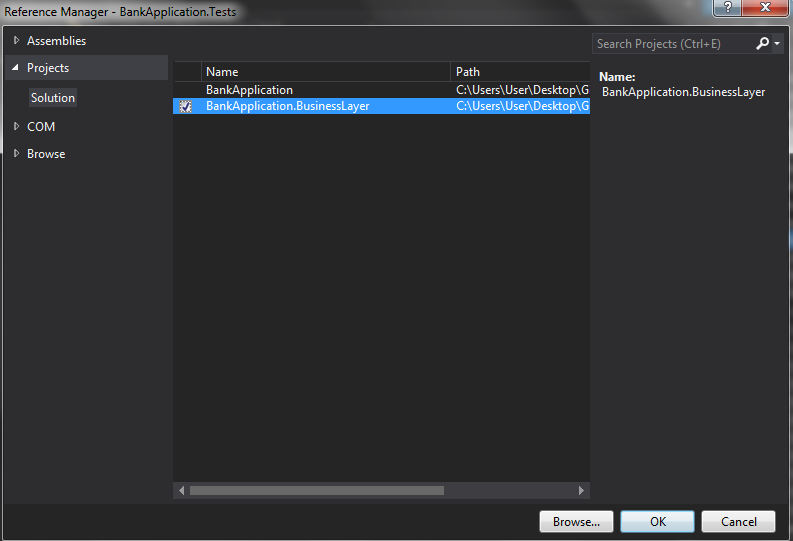


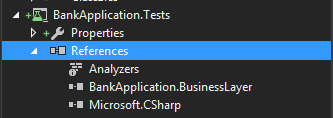


Let’s change our references in the Test project and replace the reference to BankApplication with a reference to BankApplication.BusinessLayer

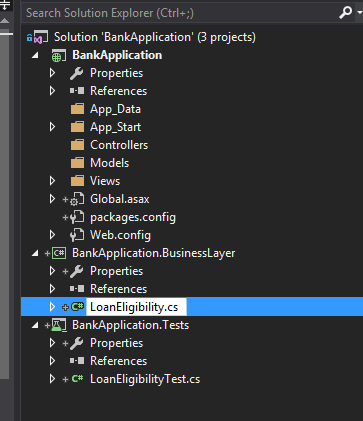


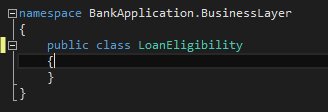






Next, let’s rename the default class in our BusinessLayer project from Class1.cs to LoanEligibility.cs





Add the method HasCorrectType to this class, with a NotImplementedException. No need to add code yet, as the current Test will pass as soon as the function is created.

public class LoanEligibility

{

public bool HasCorrectType(string loanType)

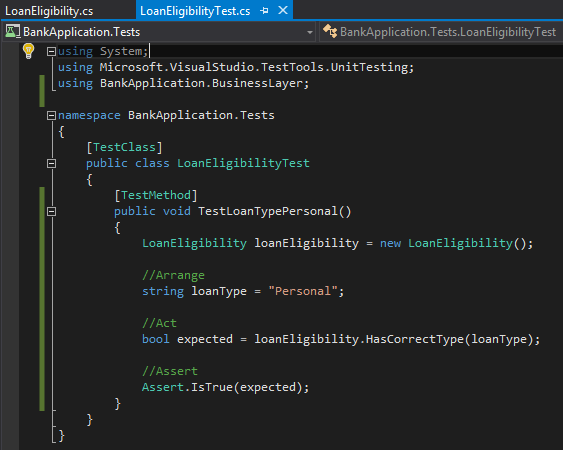
{

throw new NotImplementedException();

}

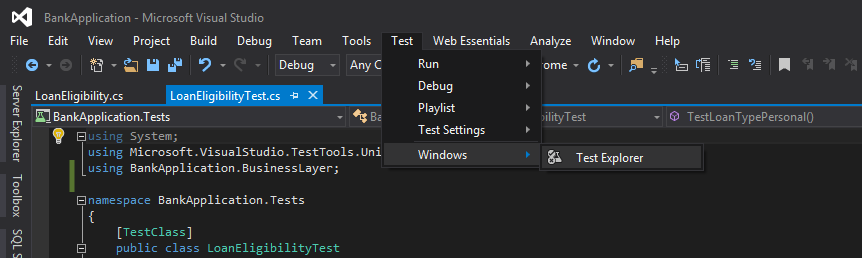
}

The compile time error in our Test goes away

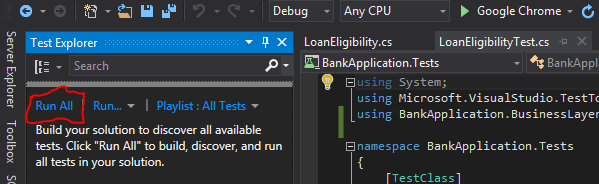


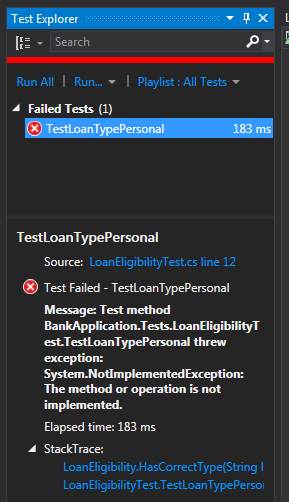
Let’s run our test now to see if it passes or fails

Go to Test 🡪 Windows 🡪 Test Explorer



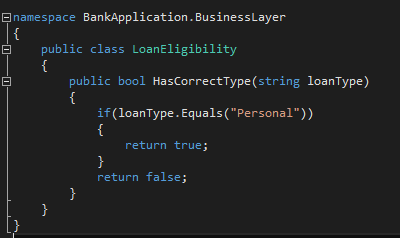
Click on Run All



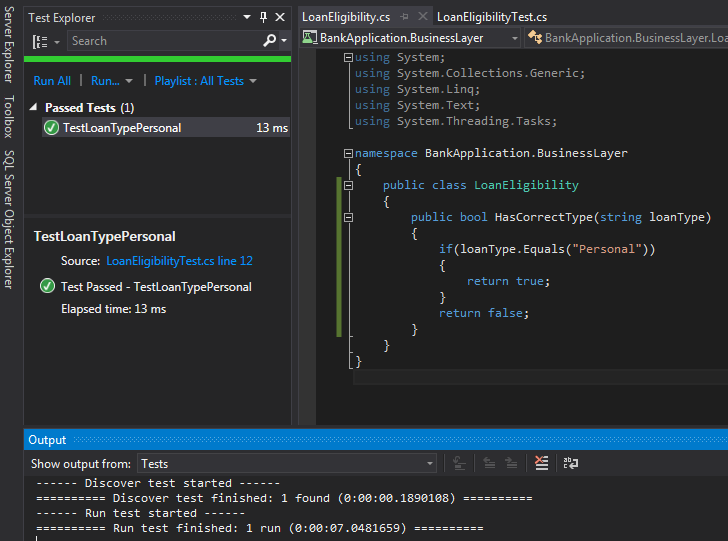


The test failed because of our NotImplementedException.

Let’s implement our method now so as to pass the test!



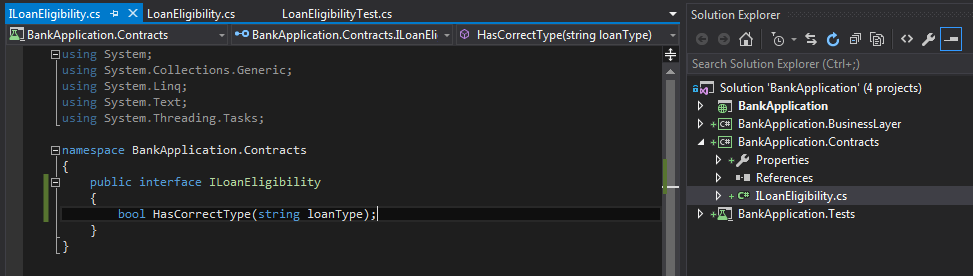
Now run the test again, it will pass this time!

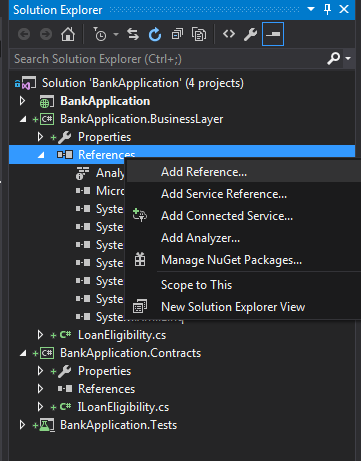


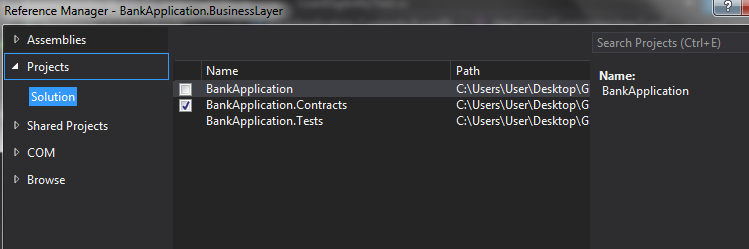
The next step is to refactor our code (both test as well as source code) if needed. This is according to the cycle: Red – Green - Refactor

Now, we can replace the actual class LoanEligibility with interface ILoanEligibility. This will decouple the tests from source code.

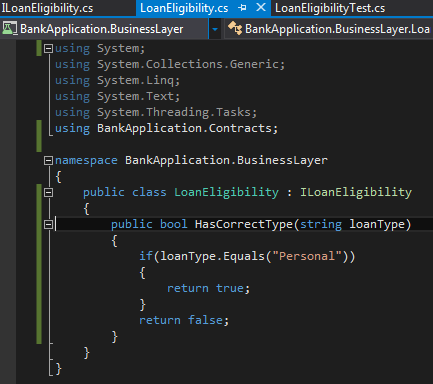
Let’s keep all our interfaces in a separate project called BankApplication.Contracts



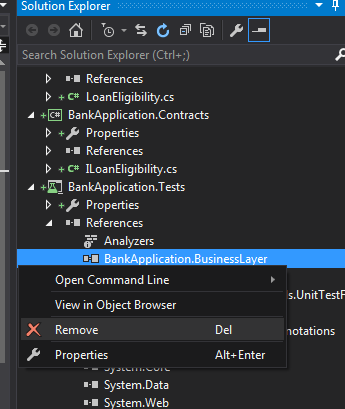


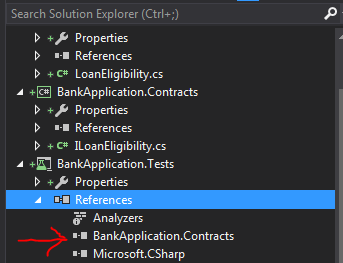


Implement ILoanEligibility in the LoanEligibility class

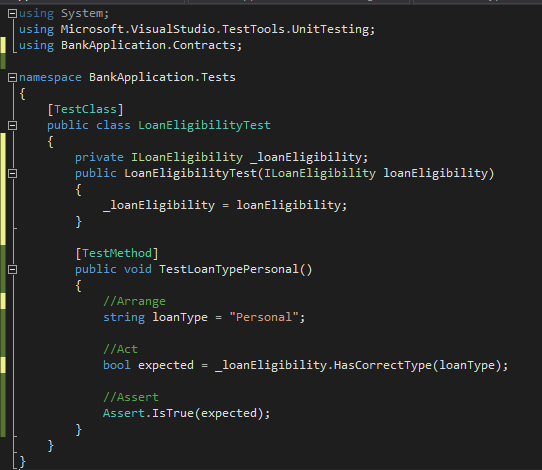


Also, in Test project, replace the reference to BankApplication.BusinessLayer with BankApplication.Contracts, and use the interface in the tests instead of the actual class





We can try to use Dependency Injection in our test cases to dynamically inject the object for ILoanEligibility, as shown below.



Notice that we have replaced actual LoanEligibility class with the interface, and we are using dependency injection to inject the actual object at the time of constructor initialization.

However, unit tests are not meant to test dependencies. That is the job of integration tests. Therefore, instead of using DI, we should use mocking frameworks to create a mock or proxy object of ILoanEligibility.

We will use the Moq library to create our mocks

