**Run the Unit Tests**

* After opening the Solution to Microsoft Visual Studio, I tried to build and I encountered 3 errors related to the **ExcelHelper\_Accessor** Class which was not existing. I then realized that the called methods are existing in **ExcelHelper** so I called them instead and I also changed the methods to public from private to access from the Unit Tests
* I then tried to Debug the **GetCellValueTest** Unit test. I found out that I had to implement the following 2 methods which I did:
  + DiagnosticsHelper.GetCallingMethodName(1)
  + ReflectionHelper.GetExecutingAssemblyFolder(Assembly.GetExecutingAssembly());

After implementing the methods, I continue to debug the unit test and to also build manually the Excel File as none had been provided. I have managed to run a successful Test.

* Next Unit Test to run was **ReadDictionaryIntoTableTest**. I also found that I had to create the necessary file to pass the test, a shortcut of this file and to implement the following method which I did:
  + TextFileIoHelper.GetShortcutTarget
* After Creating the test file for the **ReadDictionaryIntoTableTest** I also managed to run the test successfully
* Last Unit Test that I start working was the **ReadTemplateBasedExcelAsDataTablesTest**. I also created a data file enough to pass the Unit Test.
* All Excel files that I created are under the **GetExecutingAssemblyFolder.**

**Refactoring**

* I noticed that in some of the methods there was a repeated code regards the State of the Sheet. I extracted a new method called **CheckSheetState**.
* **ReadDictionaryIntoTable**
  + There is an unnecessary *rowCount* variable which I replaced by using the Skip() method to avoid reading the 1st row
  + There was an unnecessary loop through the cells of each row and a statement to prevent reading after the 1st column. I refactored the second loop.
* **ReadTemplateBasedExcelAsDataTables**
  + I moved the validation of the first row outside the row loop so now the loop will only happen if the 1st row is valid.
  + I have replaced repeated code again for reading the data of the 1st row.
* **MapFastLenderLtdChargeType**
  + It is good practice to collect all the constant variables to a separate Constants file. I understand that Fast Lender Ltd requires a re-implementation of most of the default methods so I have also created a separate **FastLenderLtdConstants** file to remove from the code the hardcode values. As an example, please see method **MapFastLenderLtdChargeType**
* **CreateStyleSheet**
  + This is the only place that some global variables are used so I have removed from the Class itself and I created local variables.
  + I then realized that we don’t need any of these as we can retrieve the attributes from the directly from workbook
* New Class Pair

I have created a new **Pair Class** out of the **IOExcelHelpers.cs** file

**Suggestions**

* Try/Catch

While I was working to build the excel files I found that the system throws exception if the file is already Open from someone else. We could avoid this behaviour by handling this exception using a try and catch statement. I have an example of into the **ReadTemplateIntoTable** method

* Logging

I have included the **Serilog** Package from the NuGet the we can use for Logging the execution of the code and any exception that this might encounter. Latest version of this package has extra functionality to write in a SQL database. Using this logger, we can create log documents that we can then analyse and find ways to improve the performance and limit the issues

* Build more unit tests

Not all the major methods have an assigned unit test. I can see a comment in **GetCellValue** method that the specified case statements have not been tested. This method be called from almost all the other methods so a unit test is something recommended.

* There are lots of unnecessary variable declarations which can be avoided. For example, in **GetNumberFormatIdFromCellStyleIndex** the code can change as below:

**From:**

var styleIndex = cell.StyleIndex.Value;

var cellFormat = cellFormats.ElementAt((int)styleIndex);

if (cellFormat == null)

return numberFormatIdValue;

**TO:**

if (cellFormats.ElementAt((int)cell.StyleIndex.Value) == null)

return numberFormatIdValue;