**Міністерство освіти і науки України**

**Національний технічний університет України**

**«Київський політехнічний інститут імені Ігоря Сікорського»**

**Факультет інформатики та обчислювальної техніки**

**Кафедра обчислювальної техніки**

**Лабораторна робота №2**

з дисципліни

«Компоненти програмної інженерії. Якість та тестування програмного забезпечення»

на тему

«Unit тестування»

Виконав:

Перевірив:

студент групи ІП-93

Бабарикін Ігор Владиславович

Домінський Валентин Олексійович

номер залікової книжки: 9311

Київ 2021

**Мета:**

Написати Unit тести з використанням методів Black Box Testing

**Завдання:**

|  |  |  |
| --- | --- | --- |
| N п/п | 9311 mod 3 | Library |
| 1 | 0 | PasswordHasher |
| 2 | 1 | BinaryFlag |
| 3 | 2 | FileWorker (any) |

Варіант = 9311 mod 3 = 2, отже провести тестування FileWorker (будь-якого)

Отже, мій вибір для лабораторної:

* .NET 5
* Бібліотека для тестування xUnit
* Бібліотека IIG.Core.FileWorkingUtils

**Хід роботи:**

**Початок роботи:**

Я створив проект “Lab2” на NET 5, додав xUnit Test Project “TestFileWorkingUtils” та бібліотеку IIG.Core.FileWorkingUtils:



Для початку Нам треба зрозуміти, як з цим працювати, тому початок роботи буде у файлі Program. Нам треба підключити простір імен бібліотеки:



Тепер спробую створити екземпляр класу FileWorker:



Бачимо, що конструктор потребує string path. З назви можна зрозуміти, що треба шлях до якогось файлу, тому я створив TempFile.txt та додав його відносний шлях до конструктору:



Тепер я хочу переглянути публічні поля/методи цього екземпляру:  


**FilePath**

Першим полем, що відноситься до Нашого екземпляру є FilePath, яке можна тільки отримати (встановити його значення можна лише при створенні):



При спробі запуску проекту видає помилку:



Отже конструктор вимагає повний шлях:



Результат запуску:



Як Ми бачимо значення поля FilePath ідентичне до того, що Ми писали у конструкторі.

Тепер спробуємо змінити поле:



Отже поле дійсно можна лише отримати.

**GetFileName**





Як можна помітити, метод може працювати без параметрів, тому виклик повинен видати Нам “TempFile.txt”:



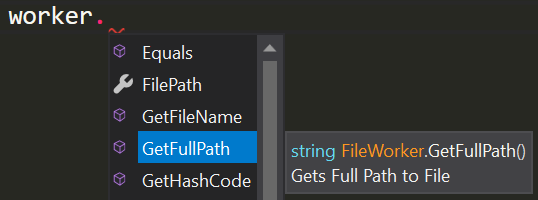
І дійсно, сталося те, про що я і казав.

Також Ми можемо викликати цей метод використовуючи клас (я дізнався про це, бо Visual Studio показує, що метод має одне додаткове перевантаження):

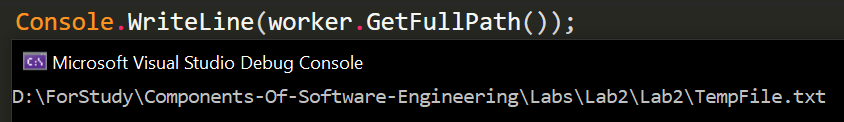




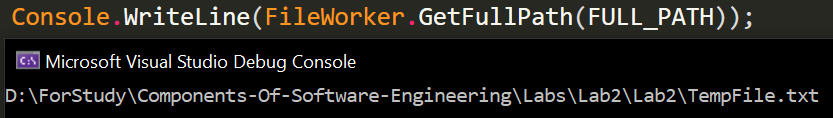
**GetFullPath**



З назви можна здогадатися, що метод повинен повертати повний шлях до файлу, який Ми вказали при створенні екземпляру. Давайте перевіримо це:

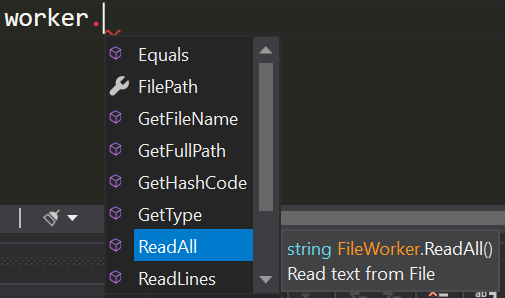


Тут теж є одне перевантаження методу, тому давайте за аналогією з минулою функцією спробуємо його викликати:

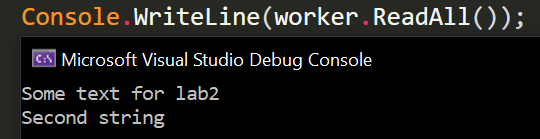
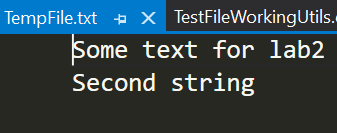


Наша теорія вірна і метод дійсно працює так само. Але викликати його не за допомогою екземпляру немає сенсу, оскільки Ми, по суті, отримуємо той самий результат, що і передали як параметри.

**ReadAll**

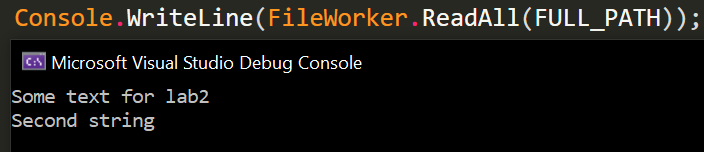


Visual Studio каже Нам, що цей метод може зчитувати дані з файлу, отже створення текстового файлу TempFile не було помилкою і Нам дійсно треба формат, який буде легко читатися.

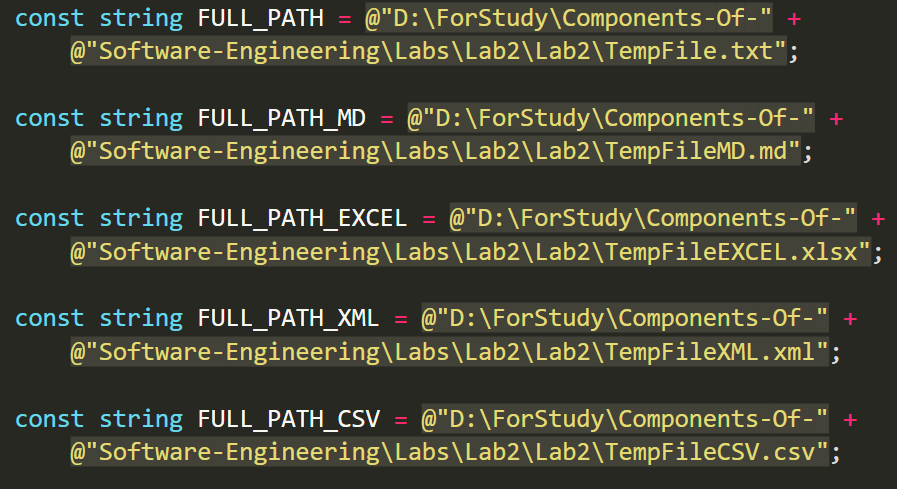
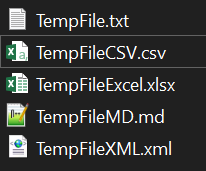
Як бачимо, метод дійсно читає усі стрічки з файлу, які за допомогою Console.WriteLine() можна вивести на екран.

Перевіримо другий варіант виклику даної функції:

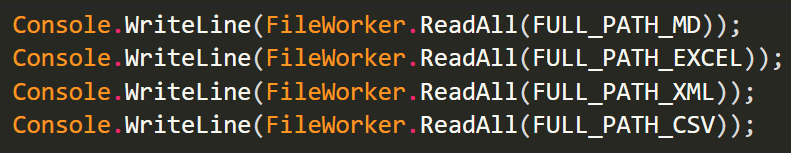


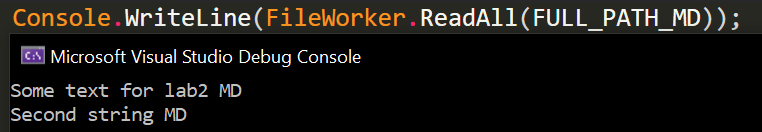
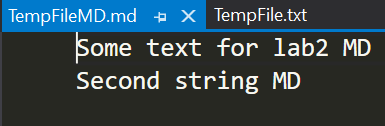
До консолі вивелися дані ідентичні до минулих, які Ми отримали використовуючи екземпляр.

Під час тестування цього методу у мене виникло питання: «А які файли можна ще відкрити?». Далі я й спробую це вияснити:

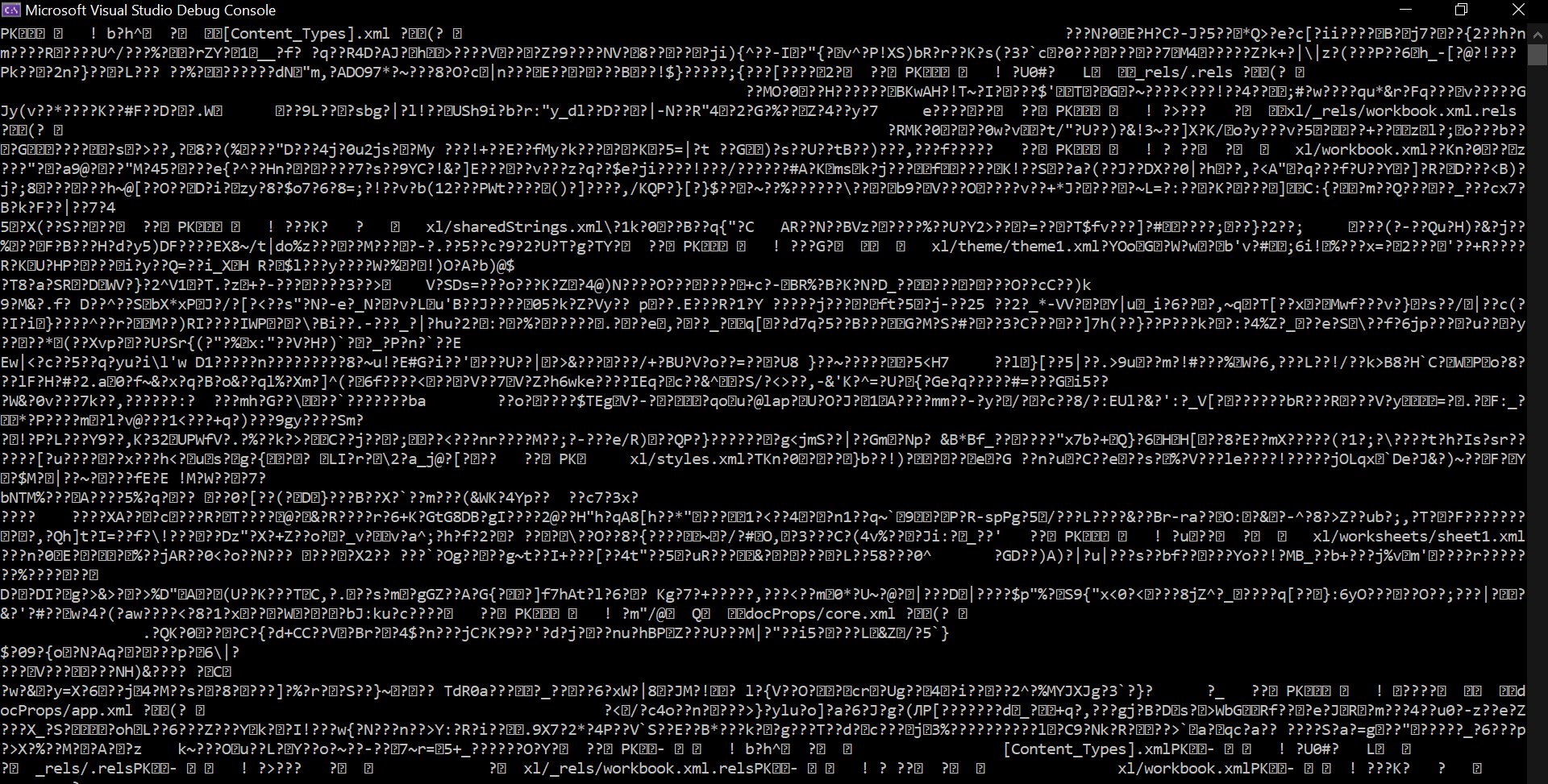
Я створив чотири додаткові файли з розширеннями \*.csv, \*.xlsx, \*.md та \*.xml, створив змінні з відповідними шляхами вивівдо консолі результати:



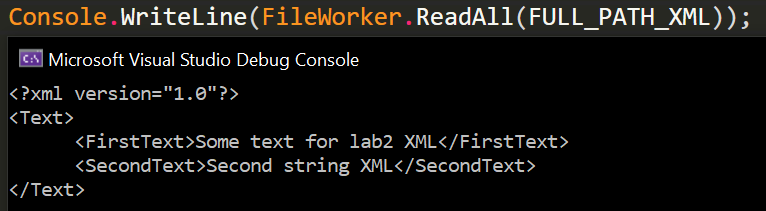
 

Метод чудово справився з форматом .\*md. Перейдемо до наступного:



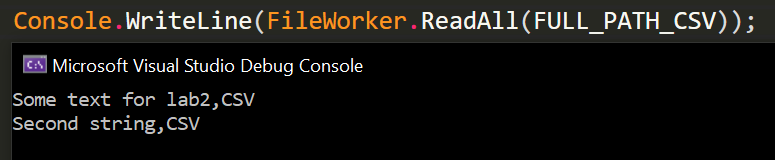


Коли я створював файл \*.xslx відразу було зрозуміло результат, оскільки для правильного читання такого формату треба доволі потужні програми, які розроблюються гігантами на кшталт Microsoft. А що там з XML?



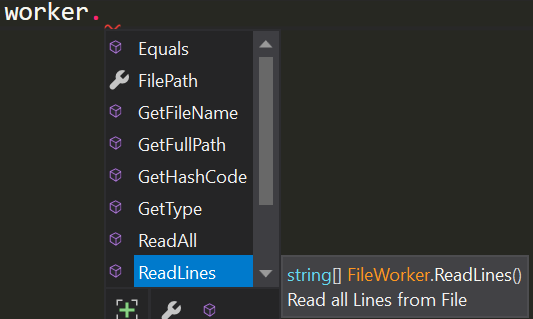
Як бачимо, файл був успішно прочитаний.

Перейдемо до \*.csv. Це, на мою думку, найцікавіший експеримент, оскільки першочергово це був документ \*.xlsx, якому змінили розширення:



Експеримент пройшов успішно. Отже, дійдемо до висновку, що файли, які потрібно відкривати спеціальними програмами (\*.xlsx) не можна успішно прочитати за допомогою цього методу. У той же час Ми маємо \*.csv файл, який при відкриванні запускається у excel. Отримуємо другий висновок: файли, які не повинні обов’язково відкриватися в окремих програмах, можна спокійно використовувати як параметри для даної функції. Ну і останнє, це те, що файли зі звичайним форматом тексту (\*.txt, \*.md, \*.xml) можна прочитати без зусиль.

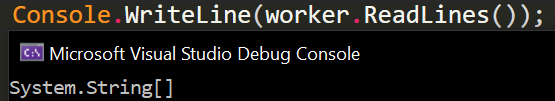
**ReadLines**



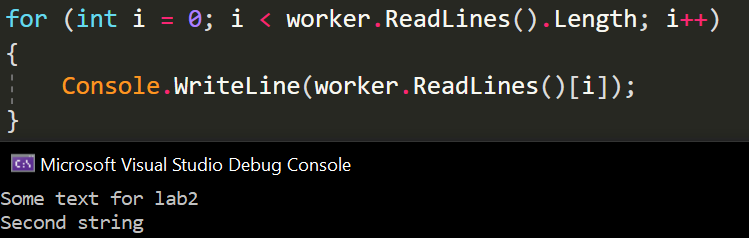
На перший погляд не зрозуміло, яка різниця між цим методом та минулим, але, якщо придивитися, то можна її помітити у значенні, яке повертається.

У даному випадку Ми отримуємо масив стрічок, на відміну від ReadAll, де була одна суцільна стрічка тексту.

Звичайний Console.WriteLine тут не підійде,

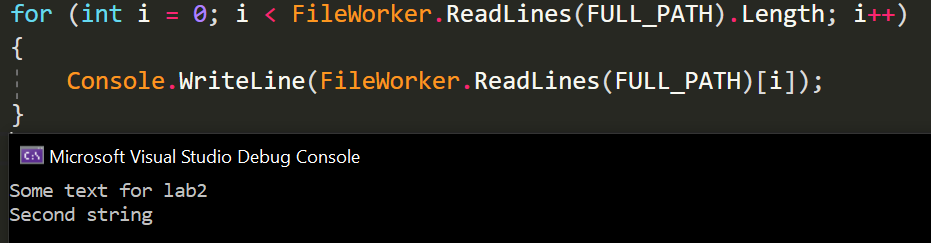


Тому треба використати цикл:



І маємо правильний вивід.

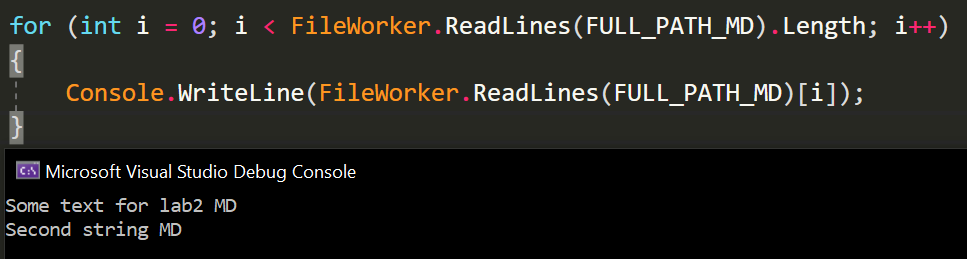
Зробимо те саме, але без екземпляру:



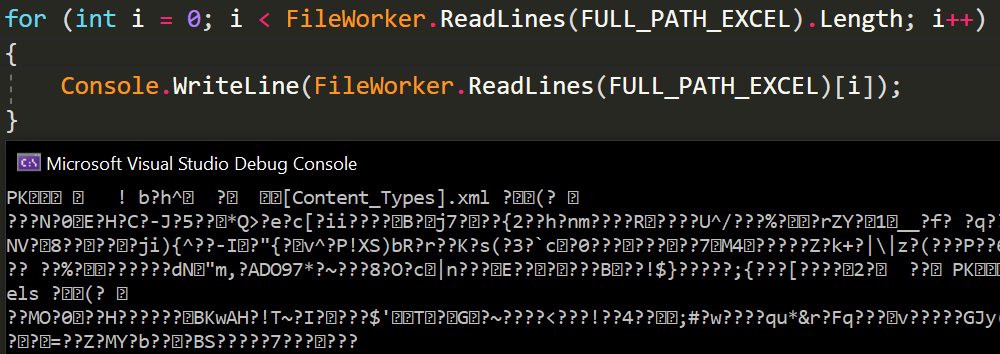
Як бачимо Стрічки ідентичні

Тепер спробуємо провернути ті самі експерименти, що й з методом ReadAll:

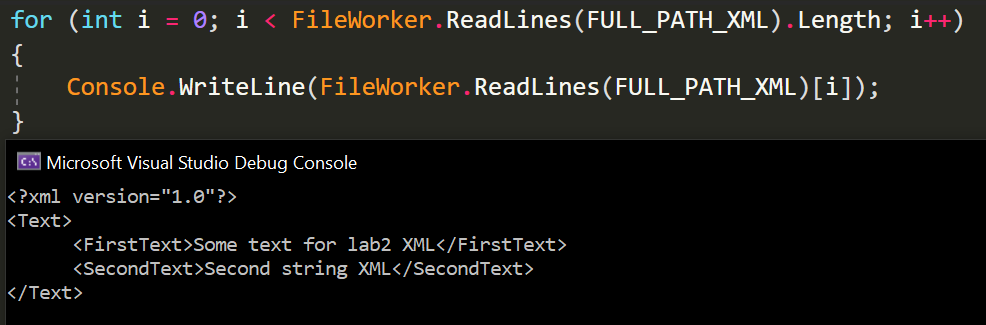
\*.md:



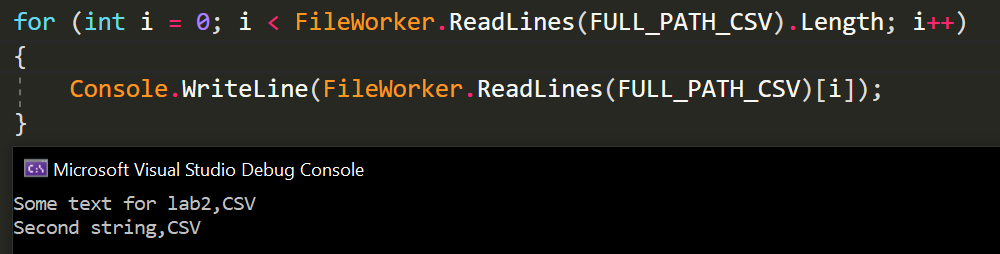
\*.xslx:



\*.xml:

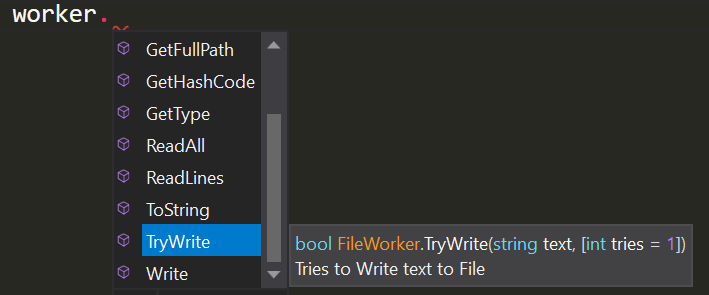


Та \*.csv:

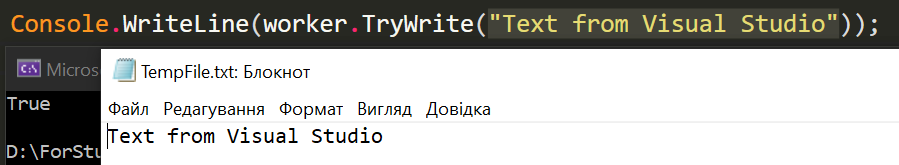


Можемо дійти до того самого висновку, що й з минулим методом.

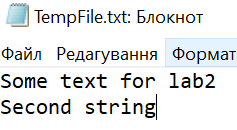
**TryToWrite**



Скоріше всього даний метод намагається лише один раз записати певну стрічку до файлу, тому давайте спробуємо це зробити:

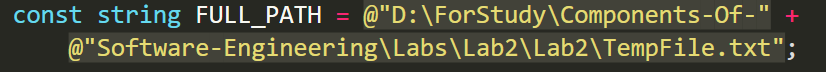


Як бачимо, минулий текст

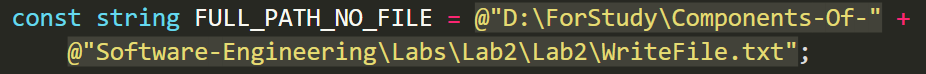


був замінений на “Text from Visual Studio”. Але ж к-сть спроб = 1, тому давайте спробуємо не дати функції записати текст до файлу:

Спочатку були спроби знайти такий символ, який не зможе розпізнатися, але дуже швидко ця ідея зійшла нанівець. Тому наступним кроком стала зміна шляху з:



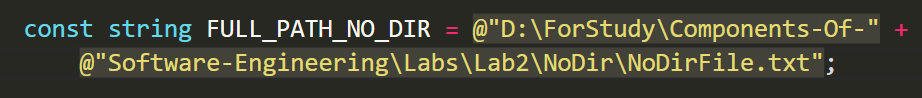
на:



Але при запуску проекту я помітив стрічку “True”. Спочатку Я не розумів, чому метод пройшов успішно, адже такого файлу просто нема. Лише через хвилину до мене дійшло, що був створений новий файл з назвою, яка задана у константі і текстом, який передається як параметр!



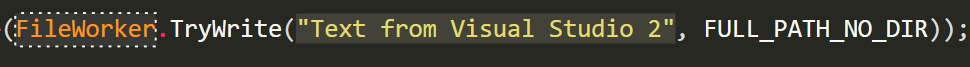
«А що Ти скажеш, якщо не буде директорії?»,- подумав я.



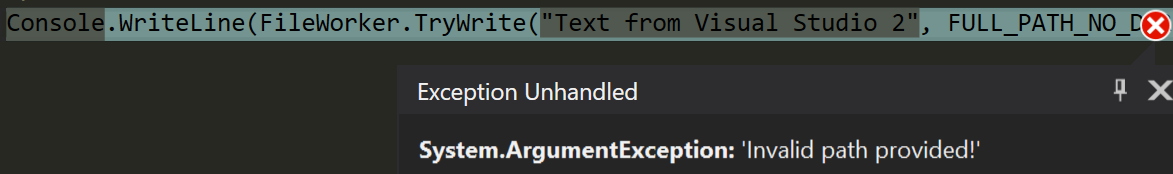
Тут мене вже зупинив компілятор, який сказав, що шлях невірний.

«Добре, я якраз не тестував для цього методу виклик через клас».

На жаль, даний варіант теж не пройшов, оскільки методу треба шлях, якого у нього нема і він ніяк не передається. Але у той же час у xml документі для бібліотеки було одне перевантаження, яке могло приймати шлях до файлу як параметр:

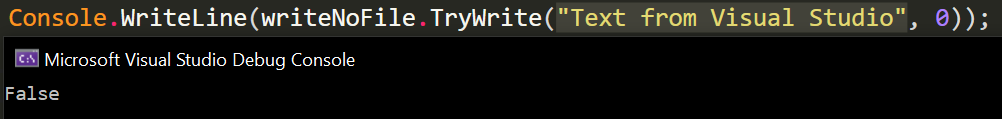


І ось, що вийшло:

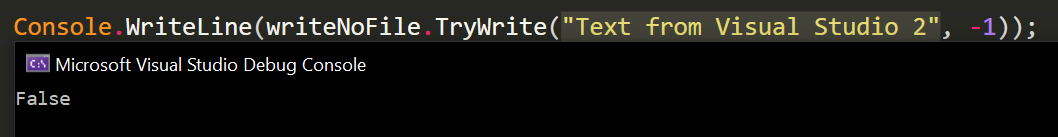


Видно, що Visual Studio скаржиться на неправильний шлях.

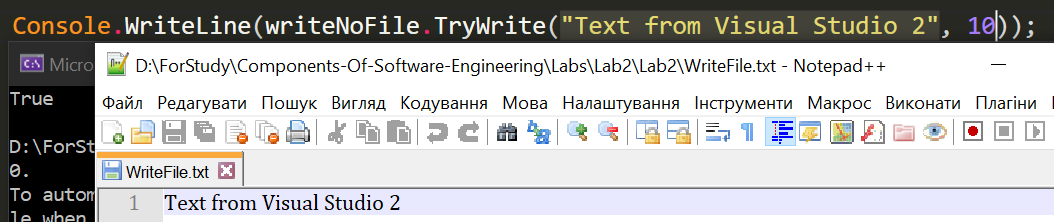
Далі я вирішив спробувати змінити кількість спроб. У першу чергу мене цікавило, що буде при значенні 0:



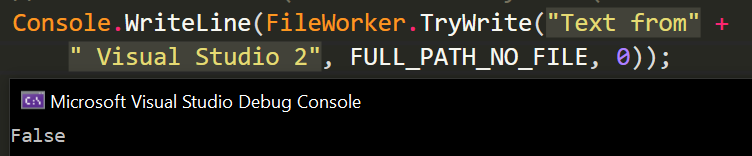
Як бачимо, метод вичерпав усі Свої спроби для запису тексту до файлу та/або створення самого документу, через що й видав Нам “False”. А як відреагує на від’ємне значення?

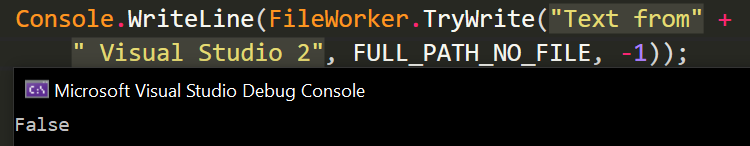


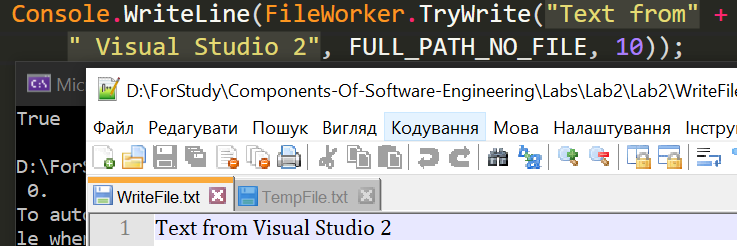
Результат не змінився. Добре, тепер спробуємо записати щось більше:



Результат показує, що функція виконалася успішно. Тепер зробимо те саме, але без екземплярів:

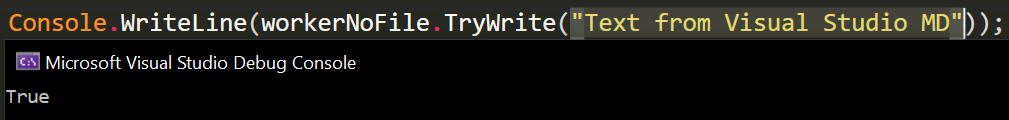




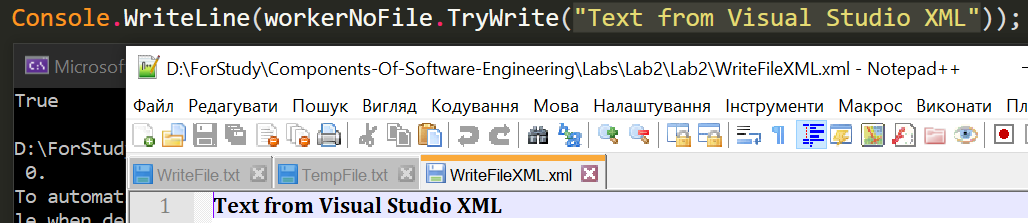


Результати виявилися ідентичними до минулих.

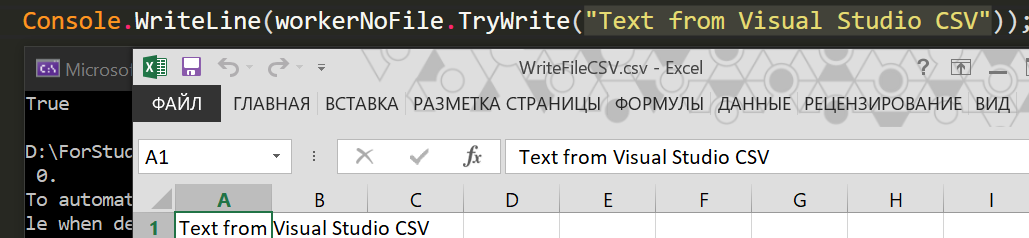
Тепер мене цікавить, які файли можна створювати за допомогою цього методу:



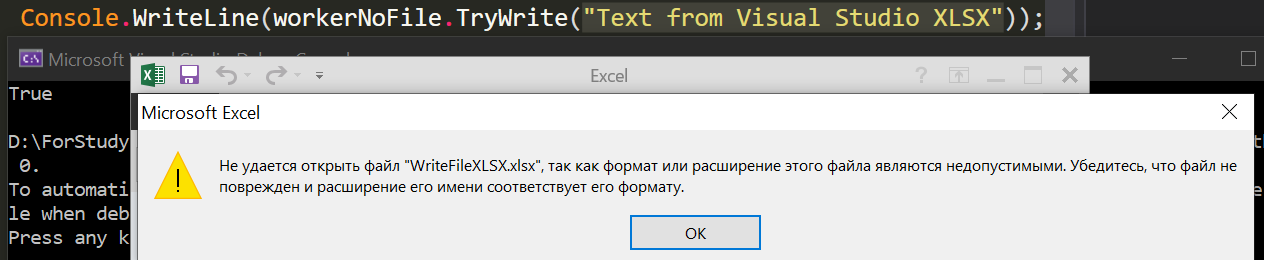
Файл \*.md пройшов.



XML теж.



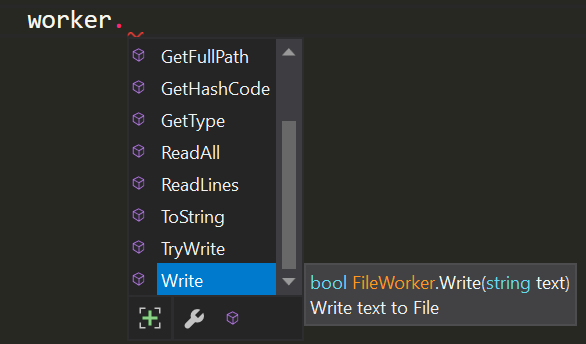
З CSV теж все успішно.



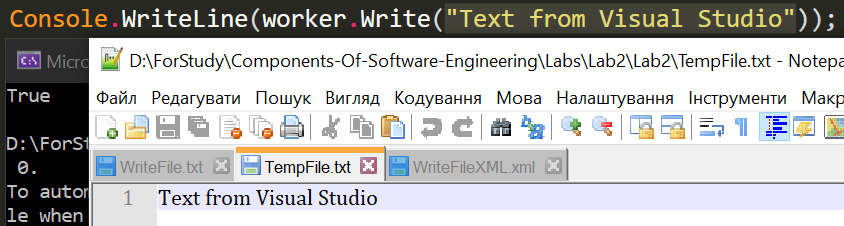
Як бачимо, файл \*.xlsx теж створився, але відкритися не може.

Отже, можемо дійти до висновку, що метод створює будь-який файл з заданим розширенням.

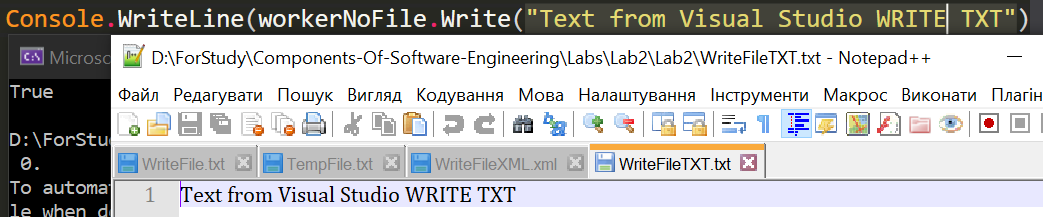
**Write**

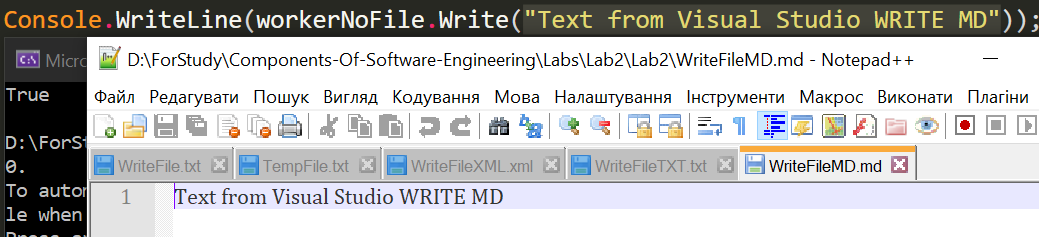


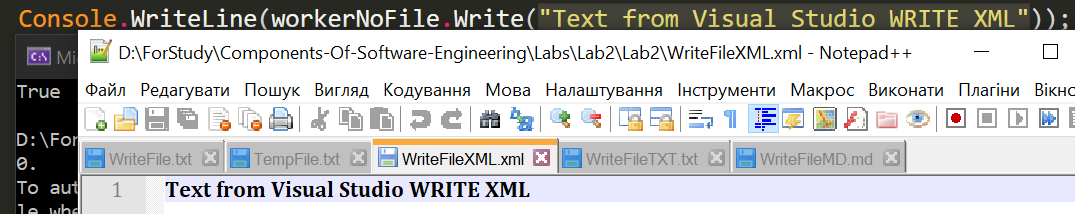
Цей метод дуже схожий на минулий, єдина різниця в тому, що спроб на запис немає.

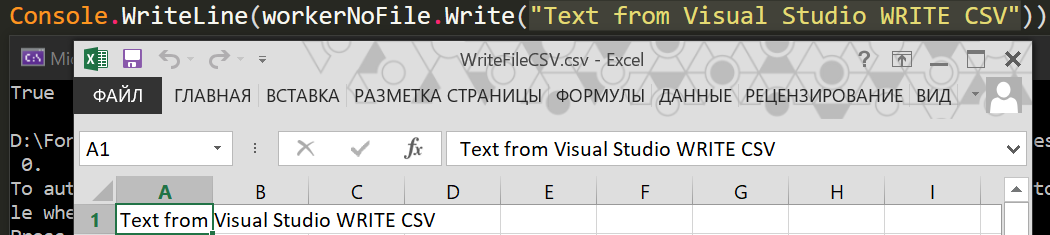


Давайте протестуємо цю команду так само, як і TryWrite():



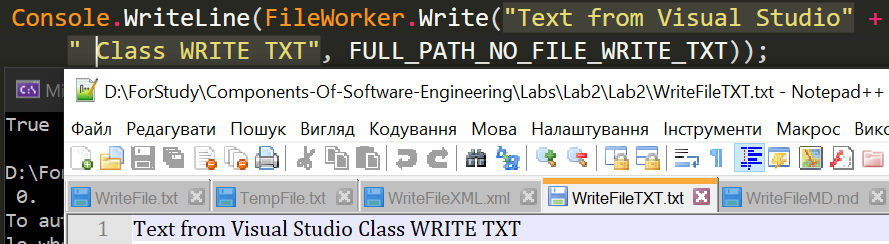


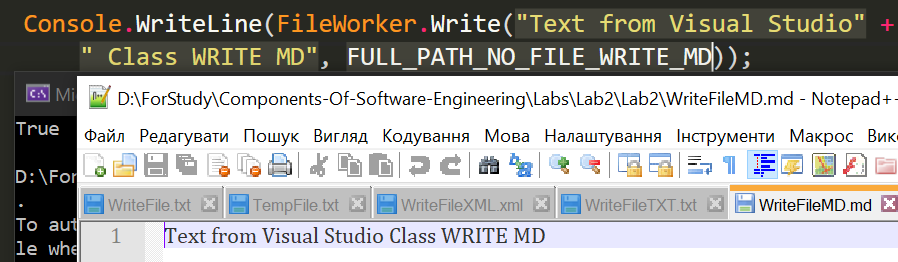


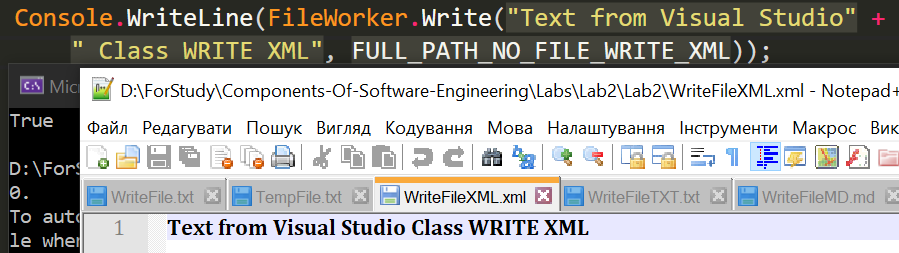


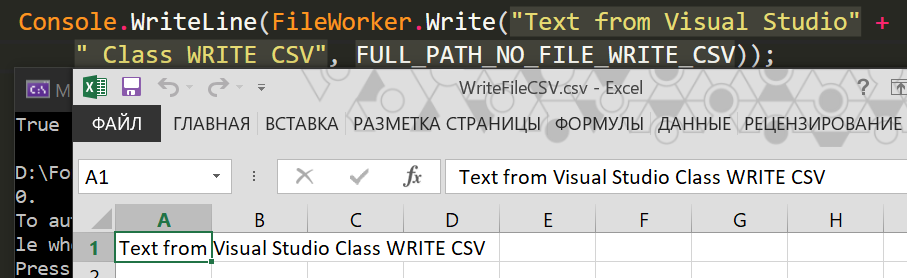
Дійсно, усе ідентично до минулої команди.

Тепер перевіримо виклик через клас:





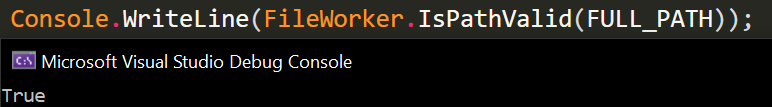




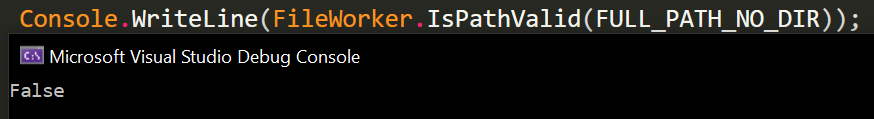
Як бачимо, усе працює.

Це всі команди, які показує Visual Studio, але в xml документі їх більше, тому давайте спробуємо їх написати.

**IsPathValid**

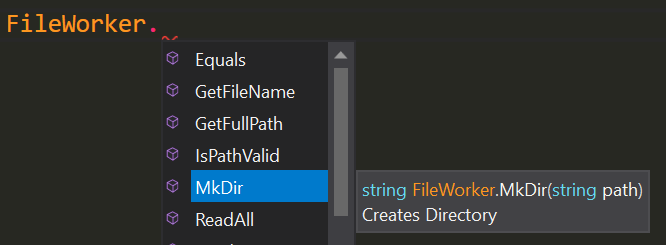


Ми записали як параметр правильний шлях і дійсно, програма розпізнала його. А тепер спробуємо задати невірний шлях:

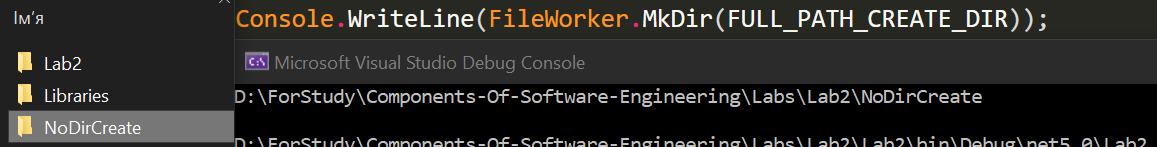


Дійсно, такого шляху не існує.

**MkDir**

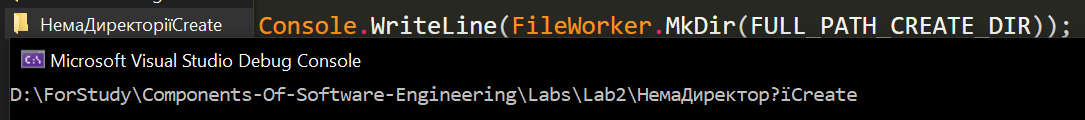


Як бачимо з опису, метод створює директорію. Давайте спробуємо:



Функція спрацювала правильно.

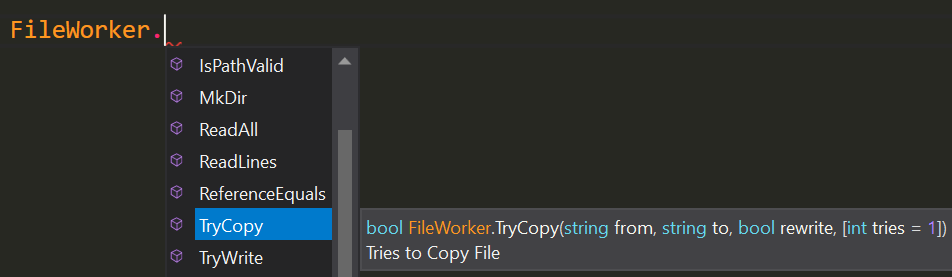
Спробуємо застосувати кирилицю:



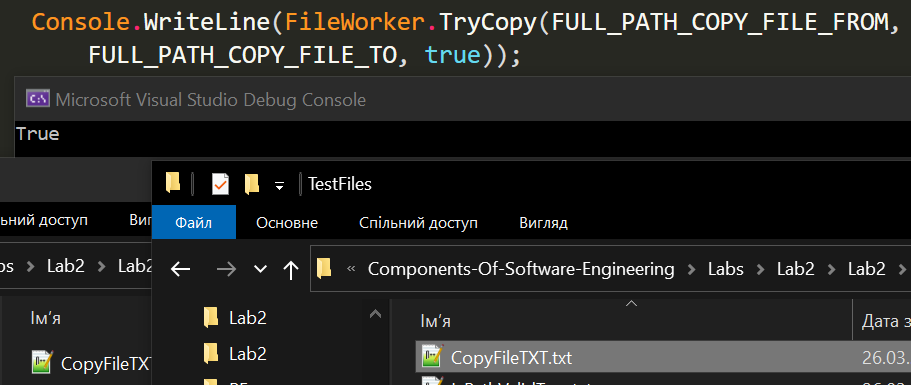
А якщо директорія вже є?

Після повторного запуску програми мені видало те саме, що і минулого разу. Метод побачив, що папка існує і тому просто видав шлях. Файли у цій директорії залишилися.

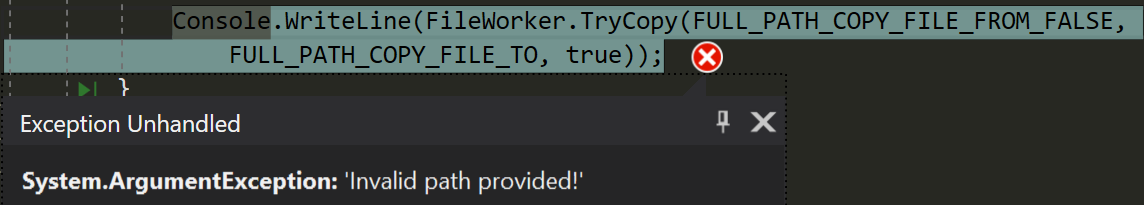
**TryCopy**

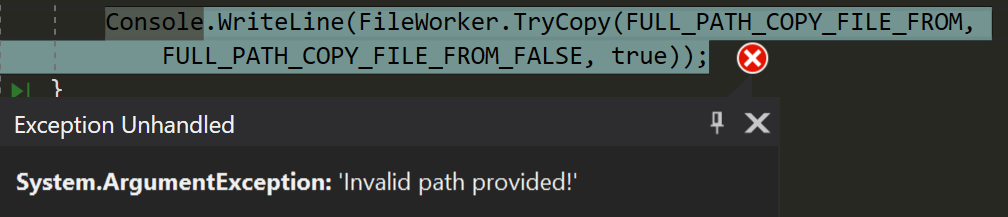
****

Як видно зі скріншоту, даний метод копіює З певного місця В іншу директорію з можливим (але не обов’язковим перезаписом).



Метод вдало виконав Свою роботу. А якщо одного з шляхів не буде існувати?





Як видно, visual studio не дає це зробити.

**Тестування:**

Тут хочу розказати деякі речі з якими я стикнувся під час тестування:

**TryCopy**

Я стикнувся з проблемою, коли при копіюванні файлу до директорії, де він уже є, виникає помилка, яку вдалося виправити лише видаленням файлу після проходження тесту.

