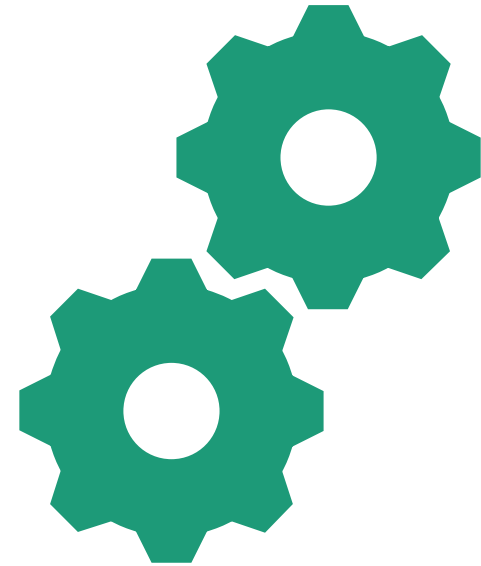


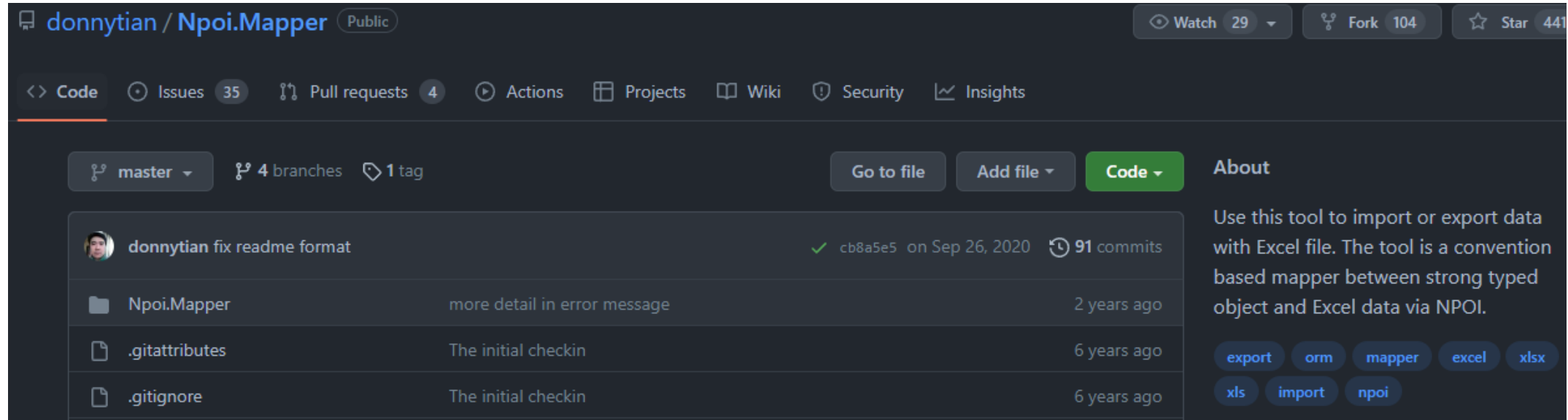
Using unit tests as a troubleshooting mechanism

By Jan Inge Nygård

20.05.2022



Repo for NPOI Mapper (excel mapper)



donnytian / Npoi.Mapper Public

Watch 29 Fork 104 Star 441

Code Issues 35 Pull requests 4 Actions Projects Wiki Security Insights

master 4 branches 1 tag

Go to file Add file Code About

Use this tool to import or export data with Excel file. The tool is a convention based mapper between strong typed object and Excel data via NPOI.

export orm mapper excel xlsx
xls import npoi

donnytian fix readme format cb8a5e5 on Sep 26, 2020 91 commits

Npoi.Mapper more detail in error message 2 years ago

.gitattributes The initial checkin 6 years ago

.gitignore The initial checkin 6 years ago

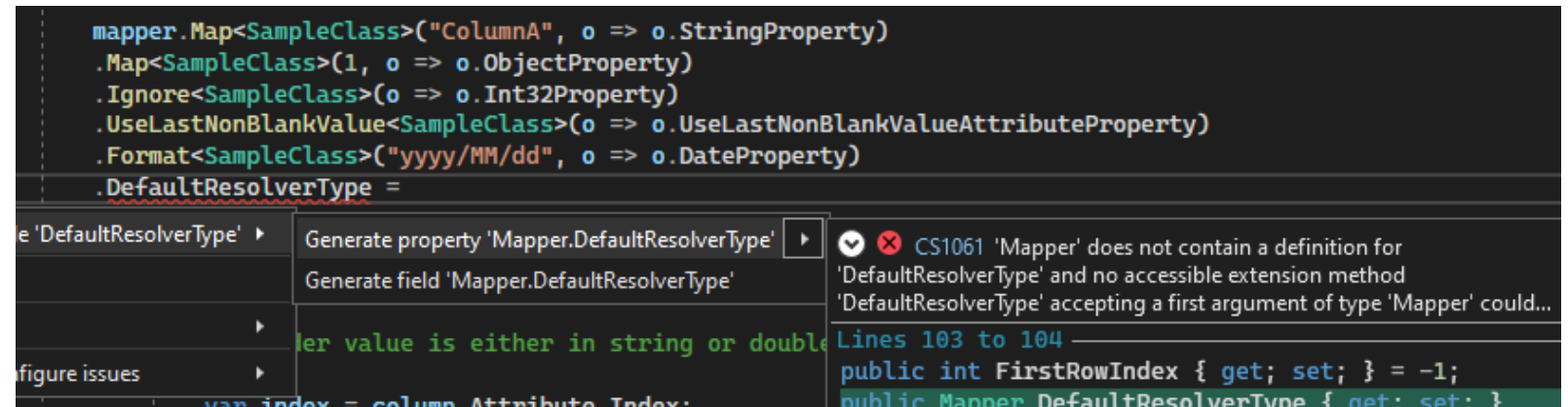
But not implemented, or removed

From Readme

Explicit column mapping

By fluent mapping methods:

```
mapper.Map<SampleClass>("ColumnA", o => o.Property1)
    .Map<SampleClass>(1, o => o.Property2)
    .Ignore<SampleClass>(o => o.Property3)
    .UseLastNonBlankValue<SampleClass>(o => o.Property1)
    .Format<SampleClass>("yyyy/MM/dd", o => o.DateProperty)
    .DefaultResolverType = typeof (SampleColumnResolver);
```



```
mapper.Map<SampleClass>("ColumnA", o => o.StringProperty)
    .Map<SampleClass>(1, o => o.ObjectProperty)
    .Ignore<SampleClass>(o => o.Int32Property)
    .UseLastNonBlankValue<SampleClass>(o => o.UseLastNonBlankValueAttributeProperty)
    .Format<SampleClass>("yyyy/MM/dd", o => o.DateProperty)
    .DefaultResolverType =
```

Generate property 'Mapper.DefaultResolverType'
Generate field 'Mapper.DefaultResolverType'

CS1061 'Mapper' does not contain a definition for 'DefaultResolverType' and no accessible extension method 'DefaultResolverType' accepting a first argument of type 'Mapper' could...

Lines 103 to 104

```
public int FirstRowIndex { get; set; } = -1;
public Mapper DefaultResolverType { get; set; }
```

Where the main test case relevant to me is

```
[Test]
public void MultiColumnContainerTest()
{
    // Arrange
    var date1 = DateTime.Now;
    var date2 = date1.AddMonths(1);
    const string str1 = "aBC";
    const string str2 = "BCD";
    const string str3 = "_PutTest";
    var workbook = GetSimpleWorkbook(date1, str1);

    // We will import columns with index of 31 and 33 into a collection property.
    workbook.GetSheetAt(1).GetRow(0).CreateCell(31).SetCellValue(date1);
    workbook.GetSheetAt(1).GetRow(0).CreateCell(33).SetCellValue(date2);

    workbook.GetSheetAt(1).GetRow(1).CreateCell(31).SetCellValue(str1);
    workbook.GetSheetAt(1).GetRow(1).CreateCell(33).SetCellValue(str2);

    // Act
    var mapper = new Mapper(workbook);
    mapper.Map(
        column => // column filter : Custom logic to determine whether or not to map and include an unmapped column.
    );
}
```

- Has Arrange, Act and Asserts already
- But some scenarios not relevant to me as well

My two main objectives

1. Map excel rows to C# model
2. Be able to have custom validation on column values (ie. null checks)

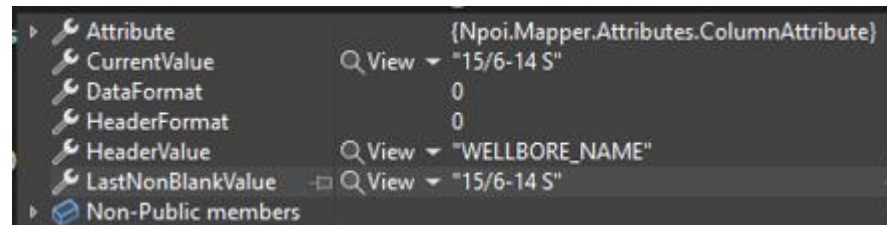
First case handled fine, with repo's attribute functionality

```
3 references | jan93, 14 hours ago | 1 author, 1 change
public partial class ExcelMapperTests
{
    4 references | jan93, 14 hours ago | 1 author, 1 change
    public class SampleClassSimple
    {
        [Column("DateProperty")]
        0 references | jan93, 14 hours ago | 1 author, 1 change
        public DateTime DateProperty { get; set; }

        [Column("StringProperty")]
        1 reference | 1/1 passing | jan93, 14 hours ago | 1 author, 1 change
        public string StringProperty { get; set; }

        [Column("OtherProperty")]
        0 references | jan93, 14 hours ago | 1 author, 1 change
        public string OtherProperty { get; set; }
    }
}
```

Maps up (behind the scenes) as expected

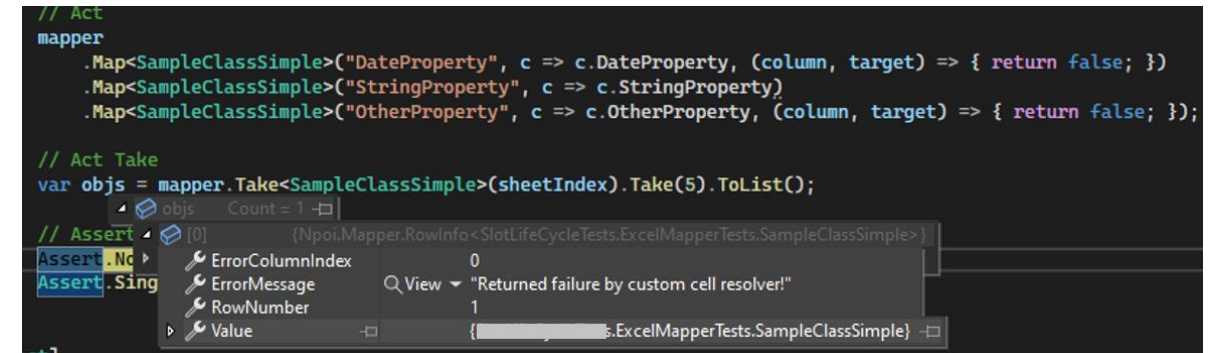


For the second case, we can also map like this

```
// Act
mapper
    .Map<SampleClassSimple>("DateProperty", c => c.DateProperty)
    .Map<SampleClassSimple>("StringProperty", c => c.StringProperty)
    .Map<SampleClassSimple>("OtherProperty", c => c.OtherProperty);

// Act Take
var objs = mapper.Take<SampleClassSimple>(sheetIndex).Take(5).ToList();
```

But only one error per row. Could also become cumbersome with multiple validations for each property



Handling custom scenarios

But for more complex scenarios, we may need more customization:

- To be able to run data validation independent of data fetching
- Especially as data fetch / load could be computationally heavy
- And for separation of concern
- This also requires overriding automapping, by setting Ignore attribute

Custom column resolver

Use overload of `Map` method to handle complex scenarios. Such as data conversion or retrieve values cross columns for a collection property.

```
mapper.Map(  
    column => // column filter : Custom logic to determine whether or not to map and include an u
```

```
/// </value>  
[Ignore] ←  
0 references | jan93, 4 hours ago |  
public string? Project
```

..but which has some consequences

Problems:

- Not all degrees of freedom covered
- Less documentation (in this case)
- Uncertainty in behaviour of application

Solution: Use unit tests to learn the application behaviour, to achieve the goal of inducing a validation error.

Approach: Start with one unit test (as close to your given scenario), and build based on that (I ended up with ~10 tests).

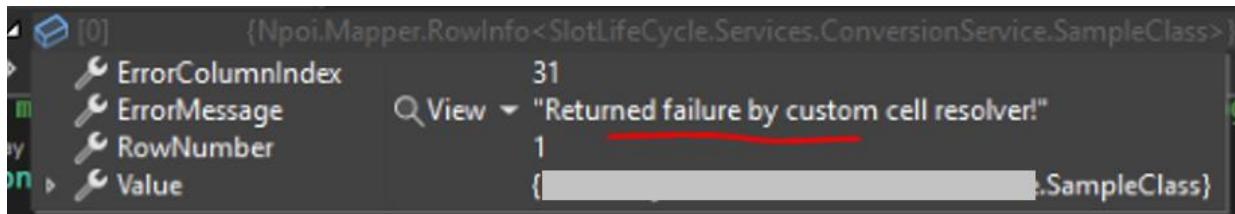
I were able to boil my scenario down to this

```
// Act
mapper.Map(
    column =>
    {
        return true;
    },
    (column, target) =>
    {
        return false;
    });
```

But I still had questions:

- Why does the error only triggers when first 'true' is returned and then 'false'?
- In lack of insight, I initially thought this was a bug

Which gives the desired error



..which for a single column looked like this

```
0 references | 0 changes | 0 authors, 0 changes
public static void PerformNullCheckForASingleColumnv0(Mapper mapper, int columnIndex)
{
    mapper.Map(
        column =>
        {
            var index = column.Attribute.Index;
            if (index == columnIndex)
            {
                return true;
            }
            return false;
        },
        (column, target) =>
        {
            if (String.IsNullOrEmpty($"{column.CurrentValue}"))
            {
                return false;
            }
            return true;
        },
        (column, source) =>
        {
            return true;
        }
    );
}
```

- While this was better, I were still a bit confused
- Why the need for a clutter of booleans?

So, I refactored with intent to create meaning

```
10 references | jan93, 9 hours ago | 1 author, 1 change
public static class CustomResolverTypes
{
    4 references | jan93, 9 hours ago | 1 author, 1 change
    public static class StageOne
    {
        public const bool IncludeColumnInNextStageValidation = true;
        public const bool IgnoreThisColumn = false;
    }

    4 references | jan93, 9 hours ago | 1 author, 1 change
    public static class StageTwo
    {
        public const bool FinalizeInMemoryMappingOperation = true;
        public const bool TriggerValidationError = false;
    }

    2 references | jan93, 9 hours ago | 1 author, 1 change
    public static class StageThree
    {
        public const bool ReturnTrue = true;
        public const bool ReturnFalse = false;
    }
}
```

- Still returning booleans
- But now with meaning
- StageThree not discovered yet, so is a placeholder to indicate next step

..which looks like this

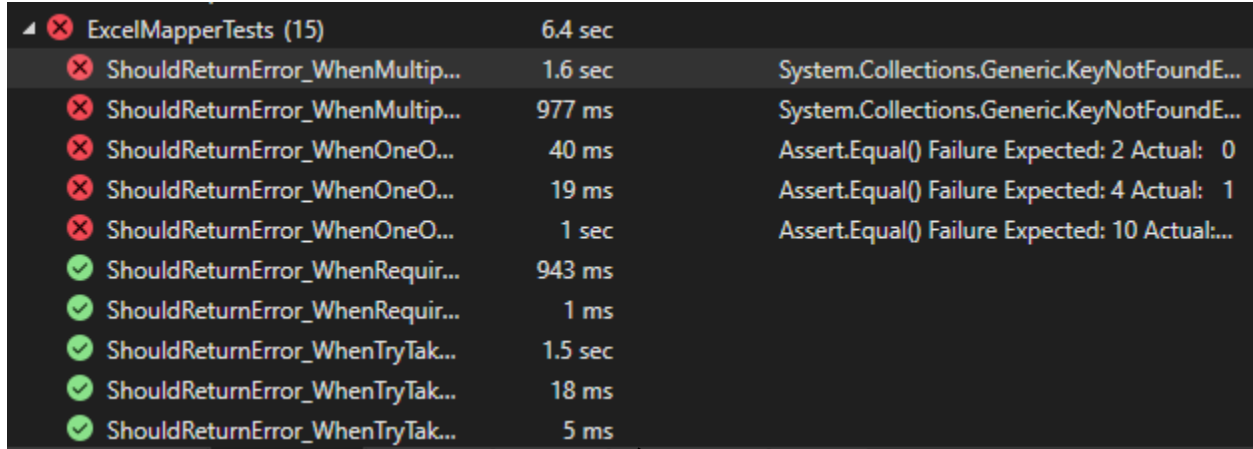
```
public static void PerformNullCheckForASingleColumn(Mapper mapper, int columnIndex)
{
    mapper.Map(
        column =>
        {
            var index = column.Attribute.Index;
            if (index == columnIndex)
            {
                return CustomResolverTypes.StageOne.IncludeColumnInNextStageValidation;
            }
            return CustomResolverTypes.StageOne.IgnoreThisColumn;
        },
        (column, target) =>
        {
            if (String.IsNullOrEmpty($"{column.CurrentValue}"))
            {
                return CustomResolverTypes.StageTwo.TriggerValidationError;
            }
            return CustomResolverTypes.StageTwo.FinalizeInMemoryMappingOperation;
        },
        (column, source) =>
        {
            return CustomResolverTypes.StageThree.ReturnTrue;
        });
}
```

- This handles validation
- But for data mapping, my idea is to do something similar

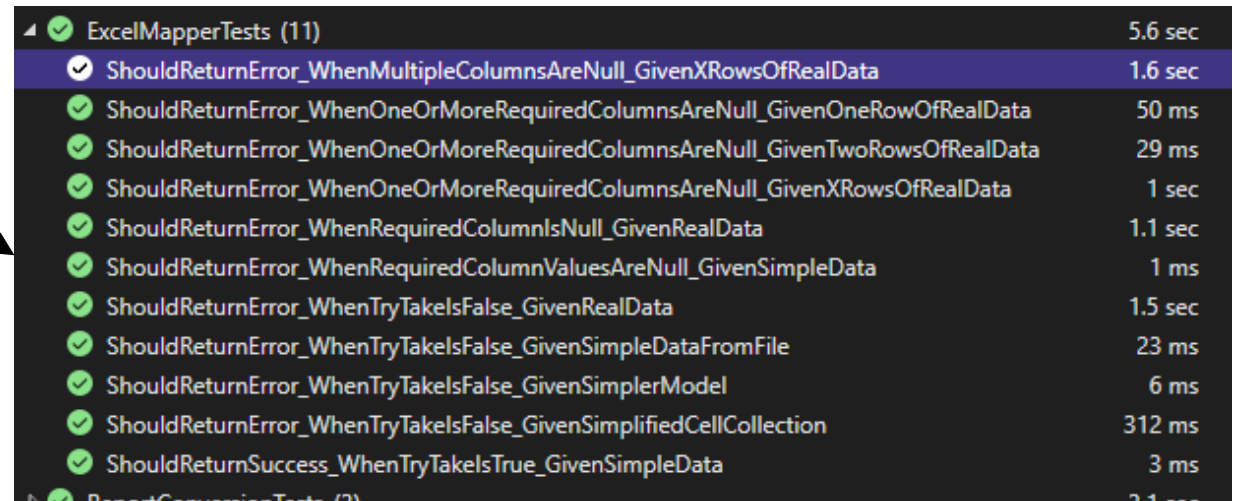
..and then, a trillion tests later



..and with some troubleshooting



ExcelMapperTests (15)	6.4 sec	
ShouldReturnError_WhenMultip...	1.6 sec	System.Collections.Generic.KeyNotFoundE...
ShouldReturnError_WhenMultip...	977 ms	System.Collections.Generic.KeyNotFoundE...
ShouldReturnError_WhenOneO...	40 ms	Assert.Equal() Failure Expected: 2 Actual: 0
ShouldReturnError_WhenOneO...	19 ms	Assert.Equal() Failure Expected: 4 Actual: 1
ShouldReturnError_WhenOneO...	1 sec	Assert.Equal() Failure Expected: 10 Actual:...
ShouldReturnError_WhenRequir...	943 ms	
ShouldReturnError_WhenRequir...	1 ms	
ShouldReturnError_WhenTryTak...	1.5 sec	
ShouldReturnError_WhenTryTak...	18 ms	
ShouldReturnError_WhenTryTak...	5 ms	



ExcelMapperTests (11)	5.6 sec	
ShouldReturnError_WhenMultipleColumnsAreNull_GivenXRowsOfRealData	1.6 sec	
ShouldReturnError_WhenOneOrMoreRequiredColumnsAreNull_GivenOneRowOfRealData	50 ms	
ShouldReturnError_WhenOneOrMoreRequiredColumnsAreNull_GivenTwoRowsOfRealData	29 ms	
ShouldReturnError_WhenOneOrMoreRequiredColumnsAreNull_GivenXRowsOfRealData	1 sec	
ShouldReturnError_WhenRequiredColumnIsNull_GivenRealData	1.1 sec	
ShouldReturnError_WhenRequiredColumnValuesAreNull_GivenSimpleData	1 ms	
ShouldReturnError_WhenTryTakesFalse_GivenRealData	1.5 sec	
ShouldReturnError_WhenTryTakesFalse_GivenSimpleDataFromFile	23 ms	
ShouldReturnError_WhenTryTakesFalse_GivenSimplerModel	6 ms	
ShouldReturnError_WhenTryTakesFalse_GivenSimplifiedCellCollection	312 ms	
ShouldReturnSuccess_WhenTryTakesTrue_GivenSimpleData	3 ms	

..we arrive at the following

- Where the test itself is more readable than before

```
[Fact]
| 0 references | jan93, 5 hours ago | 1 author, 1 change
public void ShouldReturnError_WhenMultipleColumnsAreNull_GivenXRowsOfRealData()
{
    // Arrange
    var sheetIndex = 0;
    var rowsToCheck = 2;
    var mustNotBeNull = RawExcelHeadersSimple.MustNotBeNull;
    var expectedErrors = mustNotBeNull.Count() * rowsToCheck;
    var workbook = WorkbookExtensions.GetWorkbookFromTestData("test.xlsx");
    var mapper = new Mapper(workbook);

    // Act
    var validationErrors = WorkbookExtensions.PerformNullChecksForMultipleColumns(mapper, mustNotBeNull, sheetIndex, rowsToCheck);

    // Assert
    Assert.Equal(expectedErrors, validationErrors.Count);
}
```

..and the main method is more declarative

- I find that these abstractions give me a new / better insight

```
public static List<RowInfo<RawExcelFormatSimple>> PerformNullChecksForMultipleColumns(Mapper mapper,
    string[] mustNotBeNull, int sheetIndex, int rowsToCheck)
{
    var rowInfos = new List<RowInfo<RawExcelFormatSimple>>();
    var rowInfoInternalDict = new Dictionary<int, List<RowInfo<RawExcelFormatSimple>>>();
    var rowIndex = 0;

    mapper.Map(
        column =>
        {
            return HandleColumnChecks(mustNotBeNull, column);
        },
        (column, target) =>
        {
            return HandleValidationChecks(column, rowInfos, rowInfoInternalDict, ref rowIndex);
        });

    mapper.Take<RawExcelFormatSimple>(sheetIndex).Take(rowsToCheck).ToList();

    AddLastRowOfValidationErrors(rowInfos, rowInfoInternalDict, rowIndex);

    var allErrors = CollectAllErrors(rowInfoInternalDict);

    return allErrors;
}
```

Learnings

- Quite useful to use unit tests, especially when making new microservices
- Unit tests are also useful as a means of understanding external packages or dependencies better, ie. if documentation is lackluster
- The "best" form of refactoring is to have better design upfront, thus avoiding having to fix or refactor design mistakes in the future



Questions?

Thanks for your time

Repo used:

- <https://github.com/donnytian/Npoi.Mapper>

Contact info:

- Email: jan.nygard@bouvet.no
- Github profile: <https://github.com/jan93>