Matthew Butt

Unit Testing the Hard Stuff

matthewbutt.com @bnathyuw

codurance
craft at heart

NB

There will be code

It will be C#

I will explain it

Axiom:

Unit Tests are Important

feedback for delivering quality software

Speed

Control

Precision

Observation:

Some projects have no Unit Tests

Why?

Why?

not Designed for Test
few Techniques
few Tools
not even many Workarounds

Unit Testing
the Easy Stuff

Designed for Test

Tools

Techniques

or at least Workarounds

.NET Web API

[Whiteboard]

Designed for Test

- Directly Executable
- Object Oriented
- Abstractions
- DI

Techniques

SOLID design

Mocking

Architecture Patterns

(MVC, ports & adapters, hexagonal…)

Tools

NUnit…

test framework

NSubstitute…

mocking framework

OWIN…

in-memory host

Workarounds

Adapters

around leaky abstractions

Mock Clock

and other system dependencies

Speed Run locally

Control Test doubles

Precision Class level

Clear purpose

Outside-in approach

Acceptance test outer loop

Unit test inner loop

Acceptance Test

In-Memory Host
Stub Externals

Controller Test

In-Memory Host

Treat as adapter

Mock Dependencies

Test interactions

Keep it thin

No domain logic

Domain Object Tests

State or Interaction?

State: stubs

Interaction: mocks

Single Responsibility

Listen to your tests!

External dependencies

Abstractions

Interfaces in Domain

Integration tests

No test if trivial

Clock

```
namespace EasyStuff.Api.Domain {
     public interface IClock {
          DateTime Now { get; }
     }
}
```

```
namespace EasyStuff.Api.Adapters {
     public class SystemClock : IClock {
          public DateTime Now => DateTime.UtcNow;
     }
}
```

```
var knownDate = new DateTime(2001, 2, 3);
var clock = Substitute.For<IClock>();
clock.Now.Returns(knownDate);
```

Unit Testing the Hard Stuff

Microsoft Azure
Data Lake Analytics

~~Financial Transactions~~

Weather Data

aberporthdata.txt

```
Aberporth
Location: 224100E 252100N, Lat 52.139 Lon -4.570, 133 metres amsl
Estimated data is marked with a * after the value.
Missing data (more than 2 days missing in month) is marked by  ---.
Sunshine data taken from an automatic Kipp & Zonen sensor marked
with a #, otherwise sunshine data taken from a Campbell Stokes
recorder.
   yyyy  mm   tmax    tmin      af    rain     sun
              degC    degC    days      mm   hours
   1970   1    7.5     3.1       7    97.5    40.2
   1970   2    6.2     1.9       7    79.2    96.2
   1970   3    6.6     2.0       8    76.1   101.2
   1970   4    8.8     4.2       2    67.0   135.2
   1970   5   14.5     8.5       0    28.1   148.9
   1970   6   18.5    11.5       0    47.3   206.5
   1970   7   16.6    11.3       0    57.4   150.3
   1970   8   17.8    12.3       0    42.6   151.8
   1970   9   16.8    11.5       0    49.4   120.4
   1970  10   13.3     8.6       0   108.3    75.4
   1970  11   10.8     6.4       0   181.4    41.7
   1970  12    7.5     3.3       6    42.9    69.4
   etc.                  etc.                 etc.
```

Extract

Transform

Load

[Whiteboard]

Not Designed for Test

Hosted in Cloud

Not Directly Executable

Hybrid Code

Closely Coupled

Few Tools

A library from MS

abstractions too leaky

Local Test Runner

this *is* useful

Few techniques

SQL unit tests

fairly gruesome

Few workarounds

Google doesn’t help

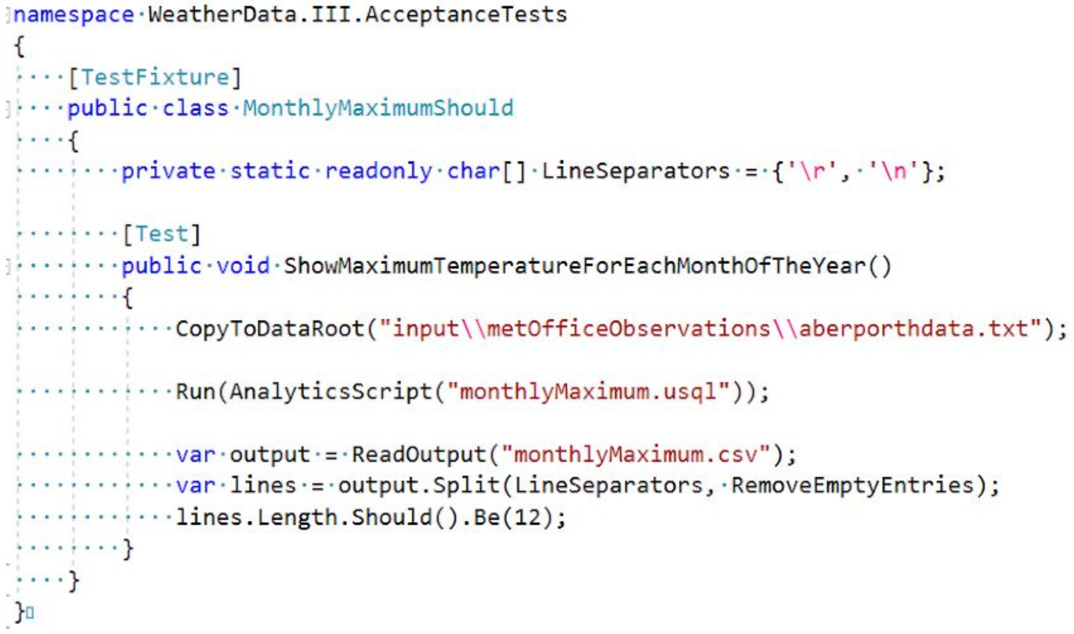
Outside-in approach

Acceptance test outer loop

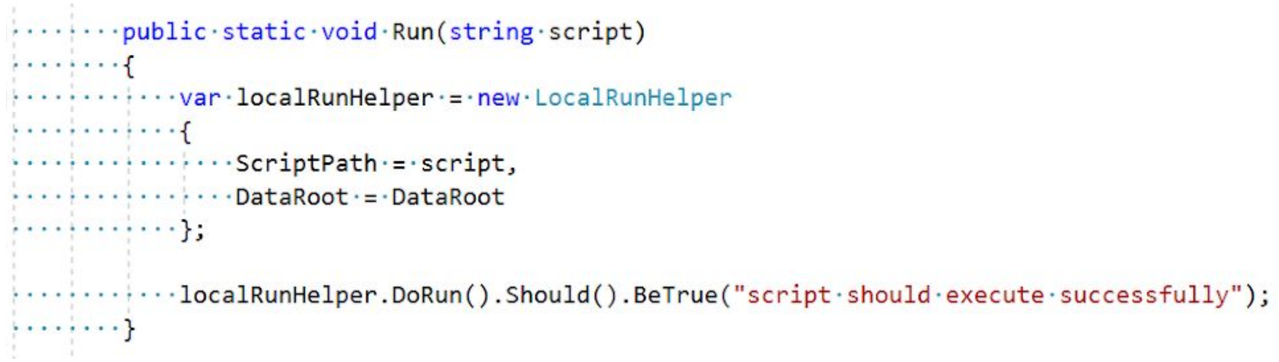
Unit test inner loop

Acceptance Test

Local Run Helper



```
namespace WeatherData.III.AcceptanceTests
{
    [TestFixture]
    public class MonthlyMaximumShould
    {
        private static readonly char[] LineSeparators = {'\r', '\n'};

        [Test]
        public void ShowMaximumTemperatureForEachMonthOfTheYear()
        {
            CopyToDataRoot("input\\metOfficeObservations\\aberporthdata.txt");

            Run(AnalyticsScript("monthlyMaximum.usql"));

            var output = ReadOutput("monthlyMaximum.csv");
            var lines = output.Split(LineSeparators, RemoveEmptyEntries);
            lines.Length.Should().Be(12);
        }
    }
}
```



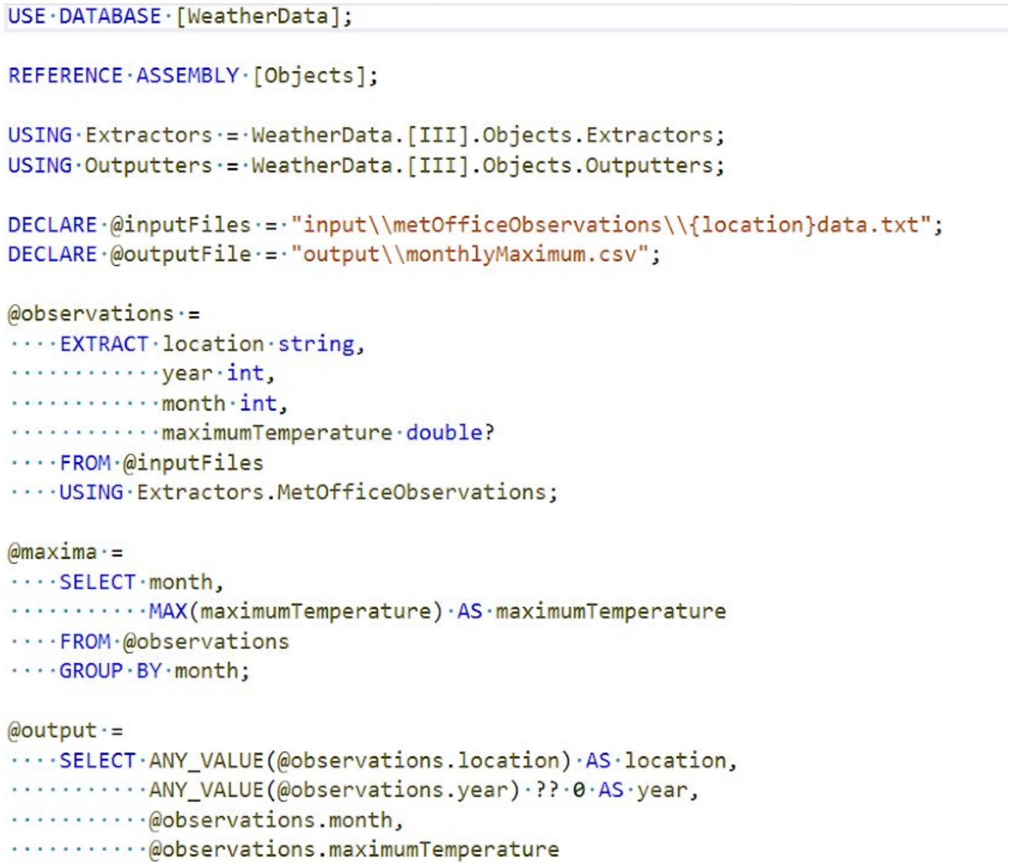
```
        public static void Run(string script)
        {
            var localRunHelper = new LocalRunHelper
            {
                ScriptPath = script,
                DataRoot = DataRoot
            };

            localRunHelper.DoRun().Should().BeTrue("script should execute successfully");
        }
```

U-SQL Script Test

Hybrid code

U-SQL & C#

Data on file system



```
USE DATABASE [WeatherData];

REFERENCE ASSEMBLY [Objects];

USING Extractors = WeatherData.[III].Objects.Extractors;
USING Outputters = WeatherData.[III].Objects.Outputters;

DECLARE @inputFiles = "input\\metOfficeObservations\\{location}data.txt";
DECLARE @outputFile = "output\\monthlyMaximum.csv";

@observations =
    EXTRACT location string,
            year int,
            month int,
            maximumTemperature double?
    FROM @inputFiles
    USING Extractors.MetOfficeObservations;

@maxima =
    SELECT month,
           MAX(maximumTemperature) AS maximumTemperature
    FROM @observations
    GROUP BY month;

@output =
    SELECT ANY_VALUE(@observations.location) AS location,
           ANY_VALUE(@observations.year) ?? 0 AS year,
           @observations.month,
           @observations.maximumTemperature
```

Responsibility of Script

Orchestrator or

Query

Seam U-SQL // C#

Inline code

Nope!

Code-behind

Nah…

Assemblies

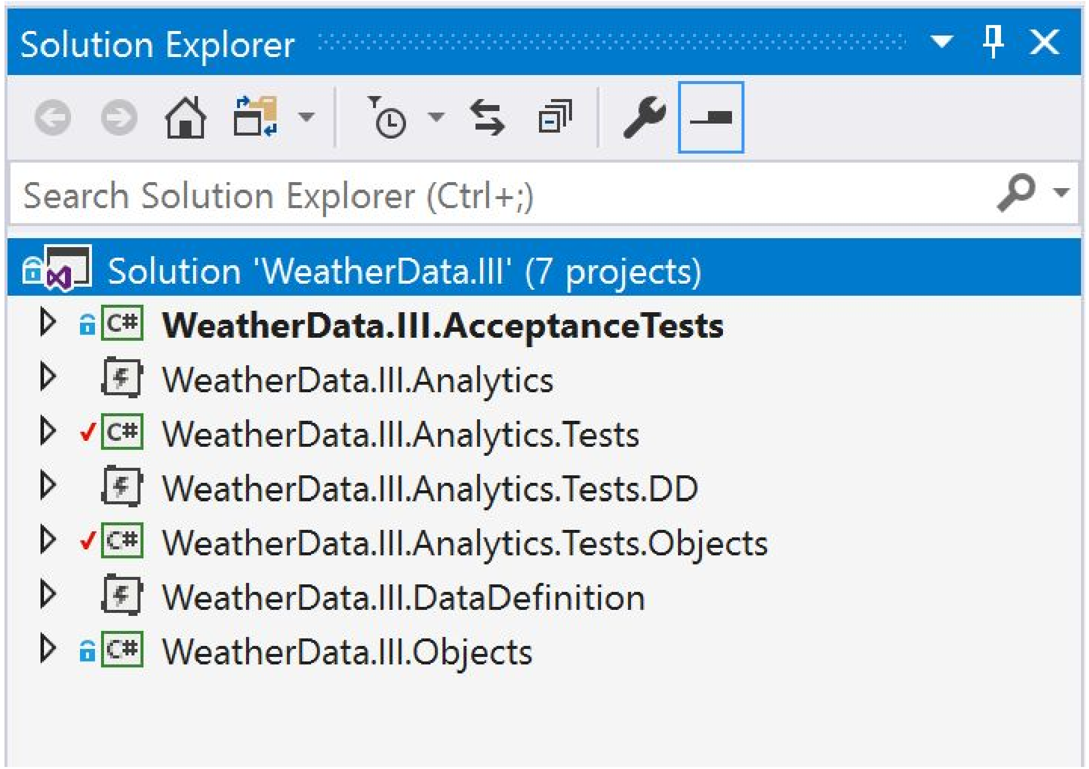
Now you’re talking!

Substitute Assemblies

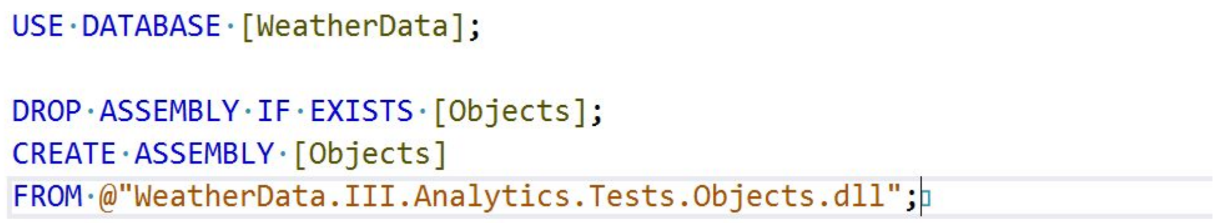
Code is compiled for each execution

Duck typing

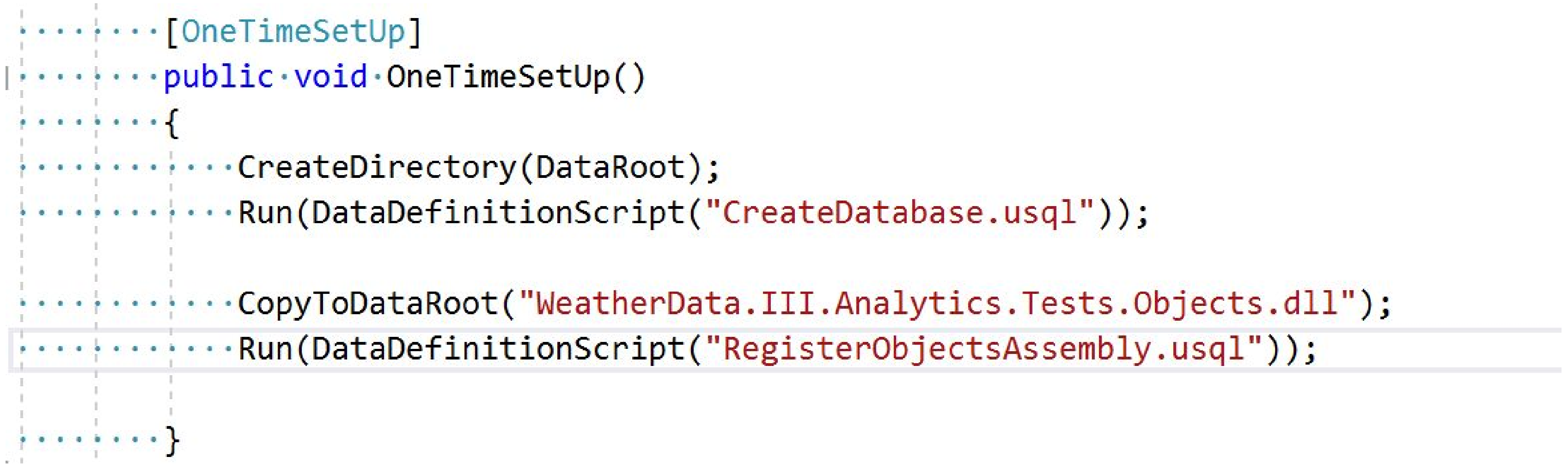
Substitute at runtime



Solution 'WeatherData.III' (7 projects)
- **WeatherData.III.AcceptanceTests**
- WeatherData.III.Analytics
- ✓ WeatherData.III.Analytics.Tests
- WeatherData.III.Analytics.Tests.DD
- ✓ WeatherData.III.Analytics.Tests.Objects
- WeatherData.III.DataDefinition
- WeatherData.III.Objects



```
USE DATABASE [WeatherData];

DROP ASSEMBLY IF EXISTS [Objects];
CREATE ASSEMBLY [Objects]
FROM @"WeatherData.III.Analytics.Tests.Objects.dll";
```

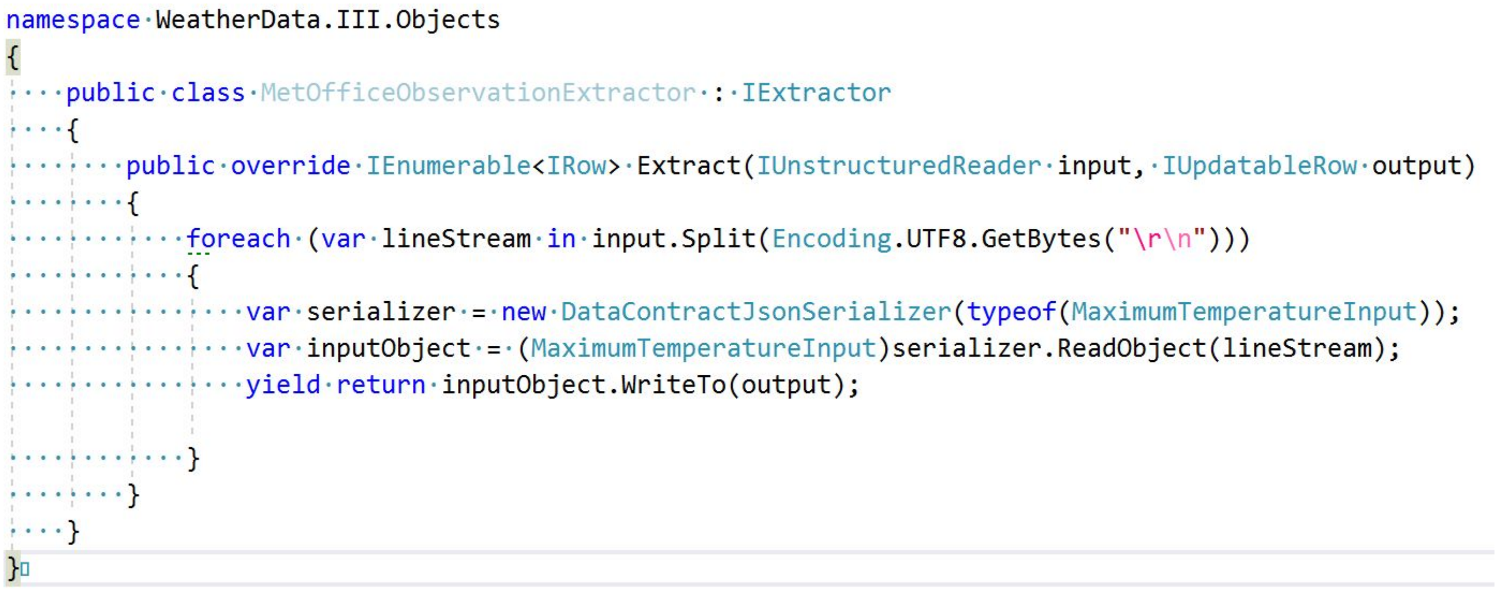


```
        [OneTimeSetUp]
        public void OneTimeSetUp()
        {
            CreateDirectory(DataRoot);
            Run(DataDefinitionScript("CreateDatabase.usql"));

            CopyToDataRoot("WeatherData.III.Analytics.Tests.Objects.dll");
            Run(DataDefinitionScript("RegisterObjectsAssembly.usql"));

        }
```

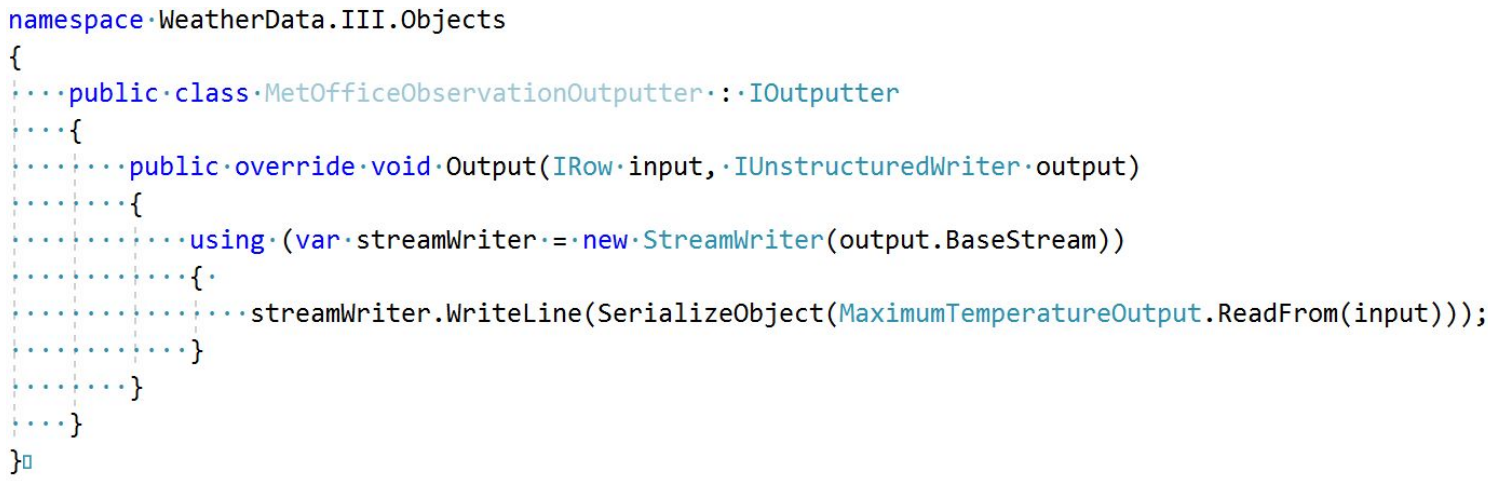
Stub or Mock?

Not directly available

Fakes against file system

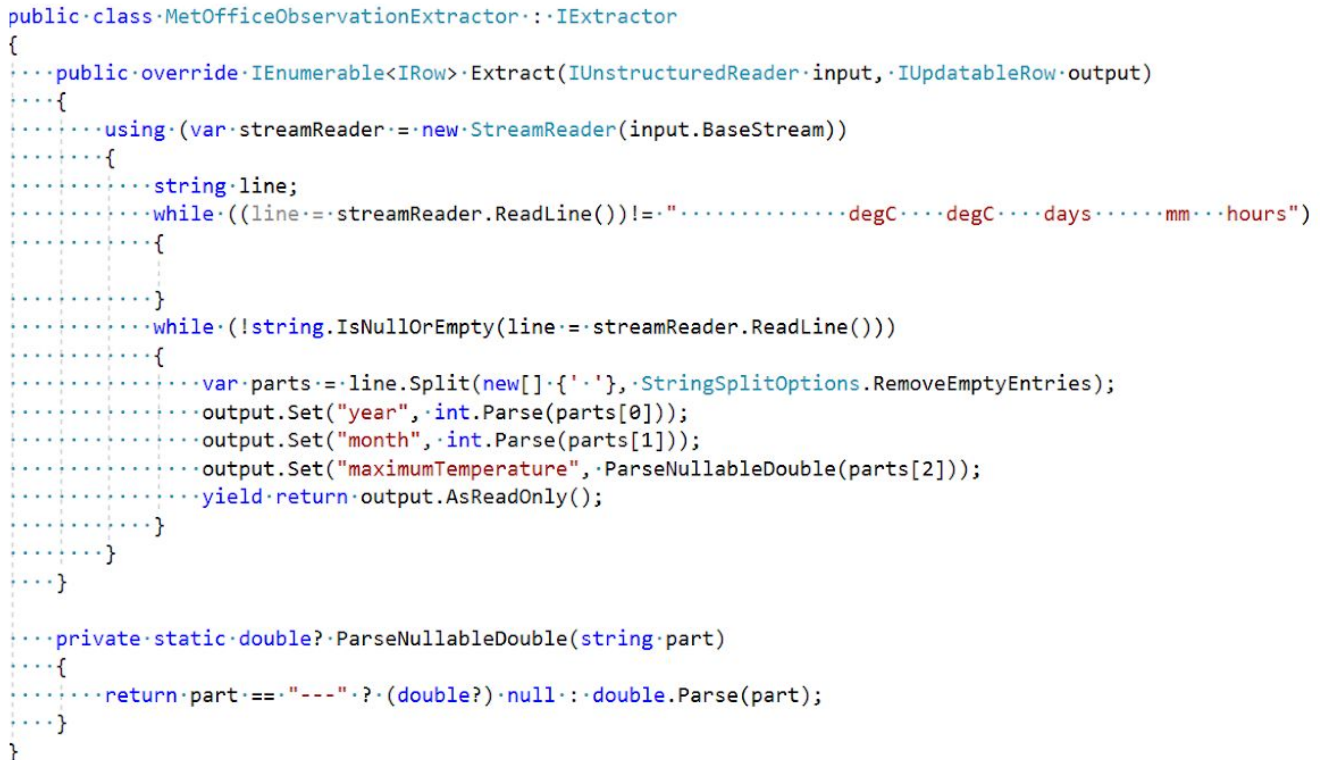


```
namespace WeatherData.III.Objects
{
    public class MetOfficeObservationExtractor : IExtractor
    {
        public override IEnumerable<IRow> Extract(IUnstructuredReader input, IUpdatableRow output)
        {
            foreach (var lineStream in input.Split(Encoding.UTF8.GetBytes("\r\n")))
            {
                var serializer = new DataContractJsonSerializer(typeof(MaximumTemperatureInput));
                var inputObject = (MaximumTemperatureInput)serializer.ReadObject(lineStream);
                yield return inputObject.WriteTo(output);

            }
        }
    }
}
```



```
namespace WeatherData.III.Objects
{
    public class MetOfficeObservationOutputter : IOutputter
    {
        public override void Output(IRow input, IUnstructuredWriter output)
        {
            using (var streamWriter = new StreamWriter(output.BaseStream))
            {
                streamWriter.WriteLine(SerializeObject(MaximumTemperatureOutput.ReadFrom(input)));
            }
        }
    }
}
```

User-Defined Objects



```csharp
public class MetOfficeObservationExtractor : IExtractor
{
    public override IEnumerable<IRow> Extract(IUnstructuredReader input, IUpdatableRow output)
    {
        using (var streamReader = new StreamReader(input.BaseStream))
        {
            string line;
            while ((line = streamReader.ReadLine())!= "              degC    degC    days      mm   hours")
            {

            }
            while (!string.IsNullOrEmpty(line = streamReader.ReadLine()))
            {
                var parts = line.Split(new[] {' '}, StringSplitOptions.RemoveEmptyEntries);
                output.Set("year", int.Parse(parts[0]));
                output.Set("month", int.Parse(parts[1]));
                output.Set("maximumTemperature", ParseNullableDouble(parts[2]));
                yield return output.AsReadOnly();
            }
        }
    }

    private static double? ParseNullableDouble(string part)
    {
        return part == "---" ? (double?) null : double.Parse(part);
    }
}
```

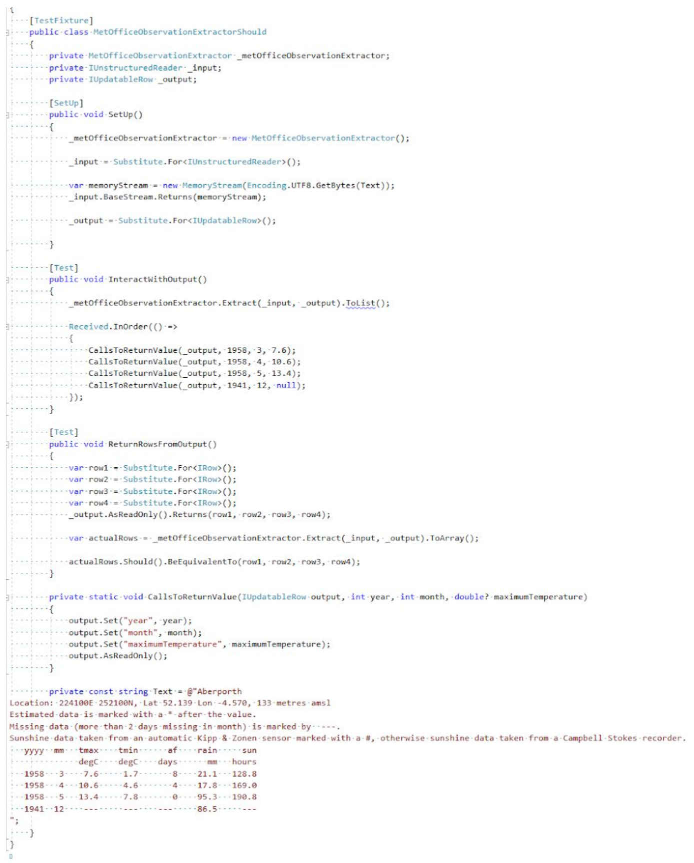
Leaky Abstraction

Use of Streams

Temporal Coupling

Strange Idiom

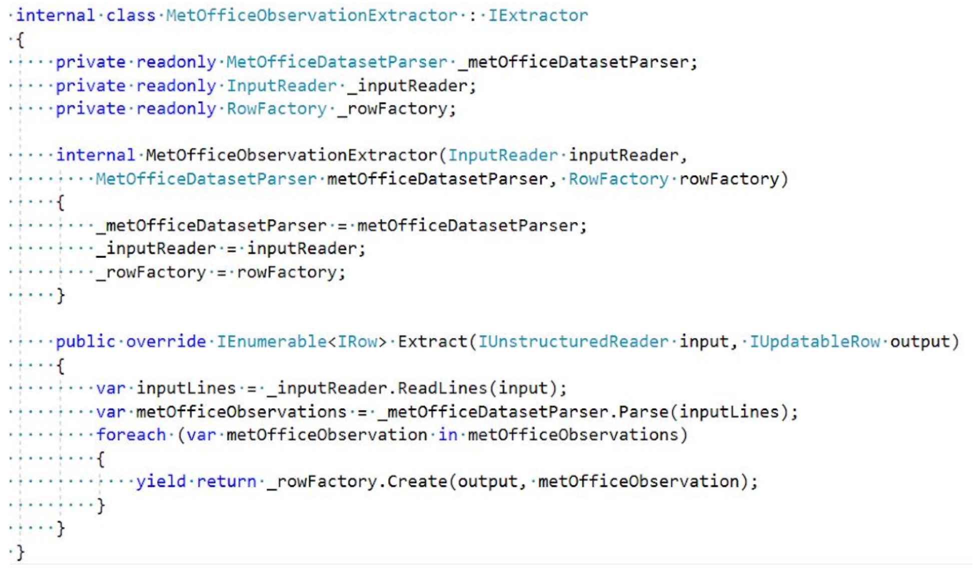
>1 Responsibility



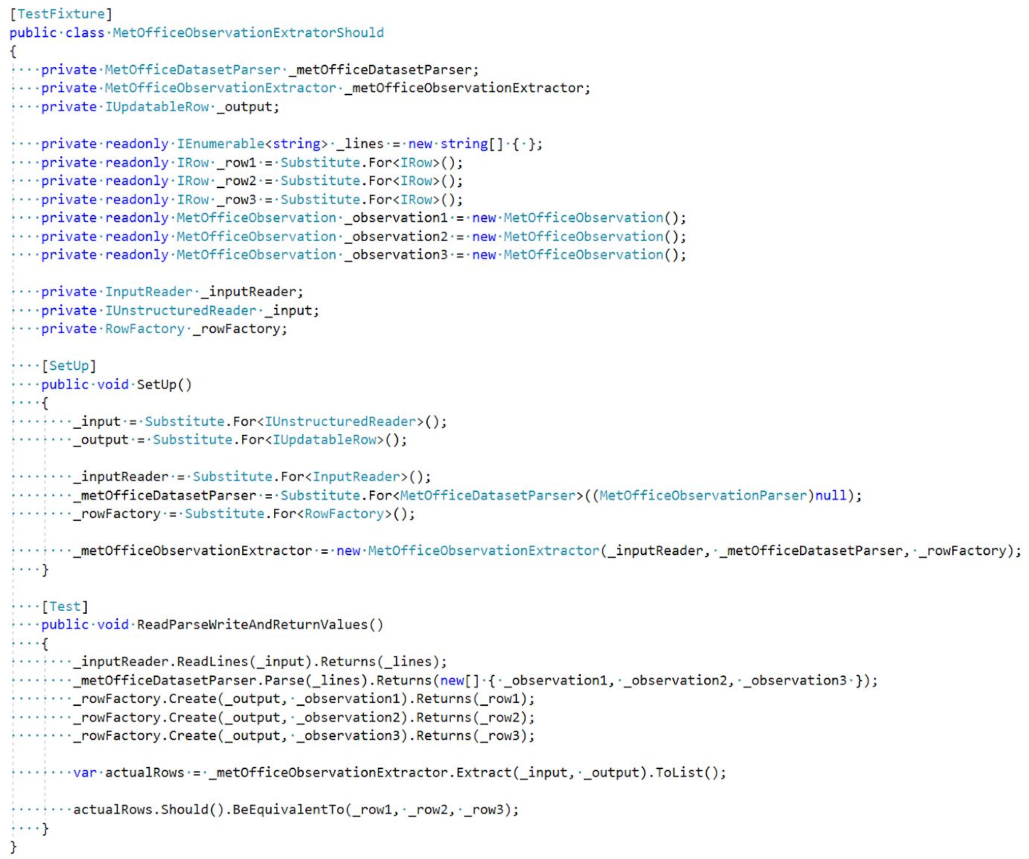
```
{
    [TestFixture]
    public class MetOfficeObservationExtractorShould
    {
        private MetOfficeObservationExtractor _metOfficeObservationExtractor;
        private IUnstructuredReader _input;
        private IUpdatableRow _output;

        [SetUp]
        public void SetUp()
        {
            _metOfficeObservationExtractor = new MetOfficeObservationExtractor();

            _input = Substitute.For<IUnstructuredReader>();

            var memoryStream = new MemoryStream(Encoding.UTF8.GetBytes(Text));
            _input.BaseStream.Returns(memoryStream);

            _output = Substitute.For<IUpdatableRow>();

        }

        [Test]
        public void InteractWithOutput()
        {
            _metOfficeObservationExtractor.Extract(_input, _output).ToList();

            Received.InOrder(() =>
            {
                CallsToReturnValue(_output, 1958, 3, 7.6);
                CallsToReturnValue(_output, 1958, 4, 10.6);
                CallsToReturnValue(_output, 1958, 5, 13.4);
                CallsToReturnValue(_output, 1941, 12, null);
            });
        }

        [Test]
        public void ReturnRowsFromOutput()
        {
            var row1 = Substitute.For<IRow>();
            var row2 = Substitute.For<IRow>();
            var row3 = Substitute.For<IRow>();
            var row4 = Substitute.For<IRow>();
            _output.AsReadOnly().Returns(row1, row2, row3, row4);

            var actualRows = _metOfficeObservationExtractor.Extract(_input, _output).ToArray();

            actualRows.Should().BeEquivalentTo(row1, row2, row3, row4);
        }

        private static void CallsToReturnValue(IUpdatableRow output, int year, int month, double? maximumTemperature)
        {
            output.Set("year", year);
            output.Set("month", month);
            output.Set("maximumTemperature", maximumTemperature);
            output.AsReadOnly();
        }

        private const string Text = @"Aberporth
Location: 224100E 252100N, Lat 52.139 Lon -4.570, 133 metres amsl
Estimated data is marked with a * after the value.
Missing data (more than 2 days missing in month) is marked by  ---.
Sunshine data taken from an automatic Kipp & Zonen sensor marked with a #, otherwise sunshine data taken from a Campbell Stokes recorder.
   yyyy  mm   tmax    tmin      af    rain     sun
              degC    degC    days      mm   hours
   1958   3    7.6     1.7       8    21.1   128.8
   1958   4   10.6     4.6       4    17.8   169.0
   1958   5   13.4     7.8       0    95.3   190.8
   1941  12    ---     ---     ---    86.5     ---
";
    }
}
```

Adapter layer

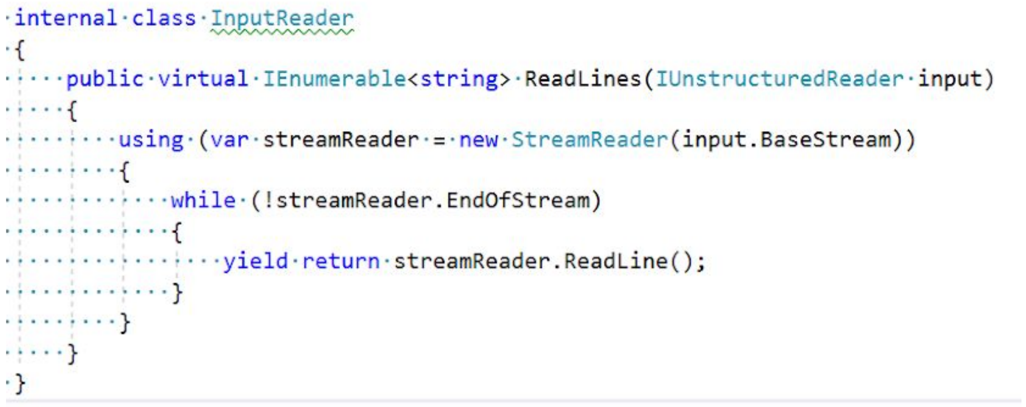
Translate between framework & domain

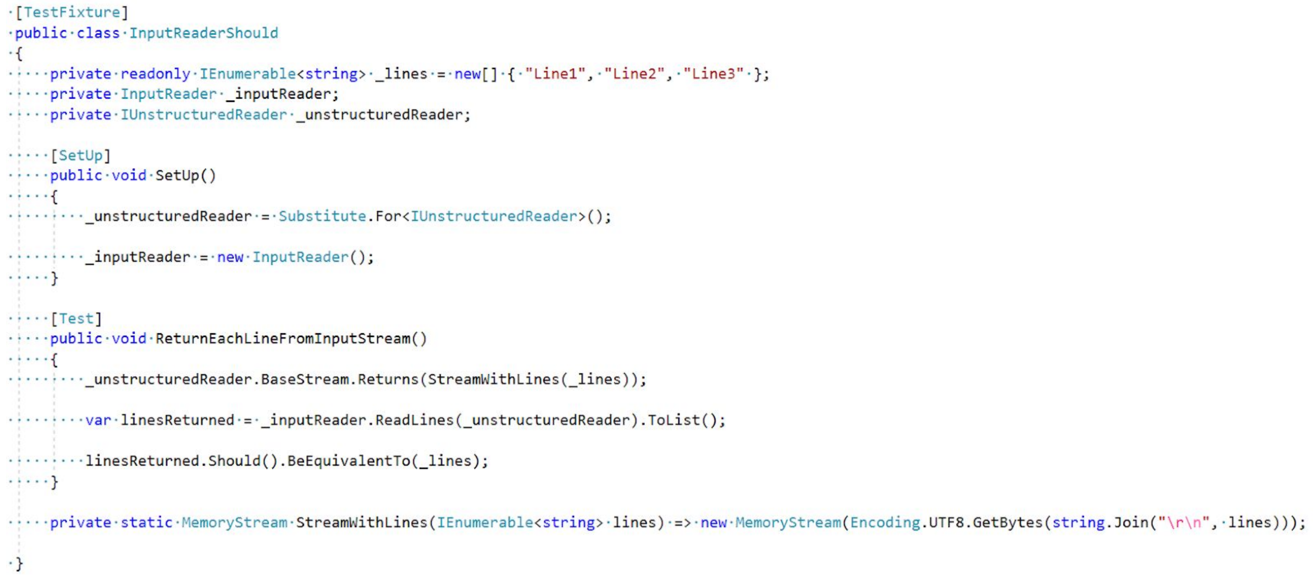


```csharp
internal class MetOfficeObservationExtractor : IExtractor
{
    private readonly MetOfficeDatasetParser _metOfficeDatasetParser;
    private readonly InputReader _inputReader;
    private readonly RowFactory _rowFactory;

    internal MetOfficeObservationExtractor(InputReader inputReader,
        MetOfficeDatasetParser metOfficeDatasetParser, RowFactory rowFactory)
    {
        _metOfficeDatasetParser = metOfficeDatasetParser;
        _inputReader = inputReader;
        _rowFactory = rowFactory;
    }

    public override IEnumerable<IRow> Extract(IUnstructuredReader input, IUpdatableRow output)
    {
        var inputLines = _inputReader.ReadLines(input);
        var metOfficeObservations = _metOfficeDatasetParser.Parse(inputLines);
        foreach (var metOfficeObservation in metOfficeObservations)
        {
            yield return _rowFactory.Create(output, metOfficeObservation);
        }
    }
}
```

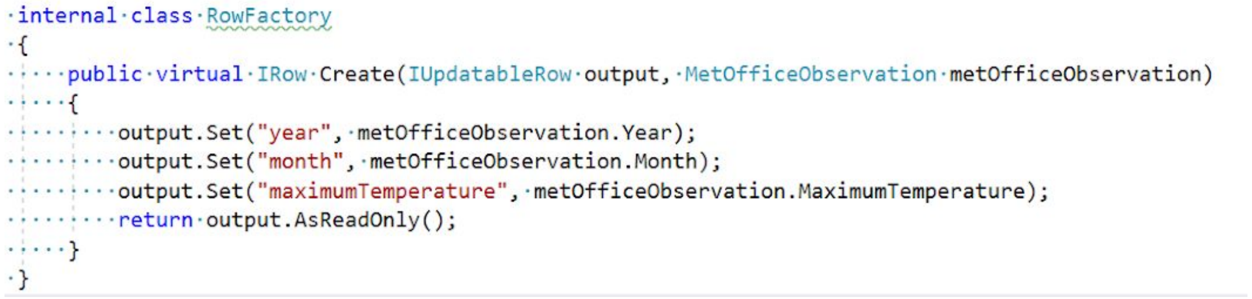


```
[TestFixture]
public class MetOfficeObservationExtratorShould
{
    private MetOfficeDatasetParser _metOfficeDatasetParser;
    private MetOfficeObservationExtractor _metOfficeObservationExtractor;
    private IUpdatableRow _output;

    private readonly IEnumerable<string> _lines = new string[] { };
    private readonly IRow _row1 = Substitute.For<IRow>();
    private readonly IRow _row2 = Substitute.For<IRow>();
    private readonly IRow _row3 = Substitute.For<IRow>();
    private readonly MetOfficeObservation _observation1 = new MetOfficeObservation();
    private readonly MetOfficeObservation _observation2 = new MetOfficeObservation();
    private readonly MetOfficeObservation _observation3 = new MetOfficeObservation();

    private InputReader _inputReader;
    private IUnstructuredReader _input;
    private RowFactory _rowFactory;

    [SetUp]
    public void SetUp()
    {
        _input = Substitute.For<IUnstructuredReader>();
        _output = Substitute.For<IUpdatableRow>();

        _inputReader = Substitute.For<InputReader>();
        _metOfficeDatasetParser = Substitute.For<MetOfficeDatasetParser>((MetOfficeObservationParser)null);
        _rowFactory = Substitute.For<RowFactory>();

        _metOfficeObservationExtractor = new MetOfficeObservationExtractor(_inputReader, _metOfficeDatasetParser, _rowFactory);
    }

    [Test]
    public void ReadParseWriteAndReturnValues()
    {
        _inputReader.ReadLines(_input).Returns(_lines);
        _metOfficeDatasetParser.Parse(_lines).Returns(new[] { _observation1, _observation2, _observation3 });
        _rowFactory.Create(_output, _observation1).Returns(_row1);
        _rowFactory.Create(_output, _observation2).Returns(_row2);
        _rowFactory.Create(_output, _observation3).Returns(_row3);

        var actualRows = _metOfficeObservationExtractor.Extract(_input, _output).ToList();

        actualRows.Should().BeEquivalentTo(_row1, _row2, _row3);
    }
}
```

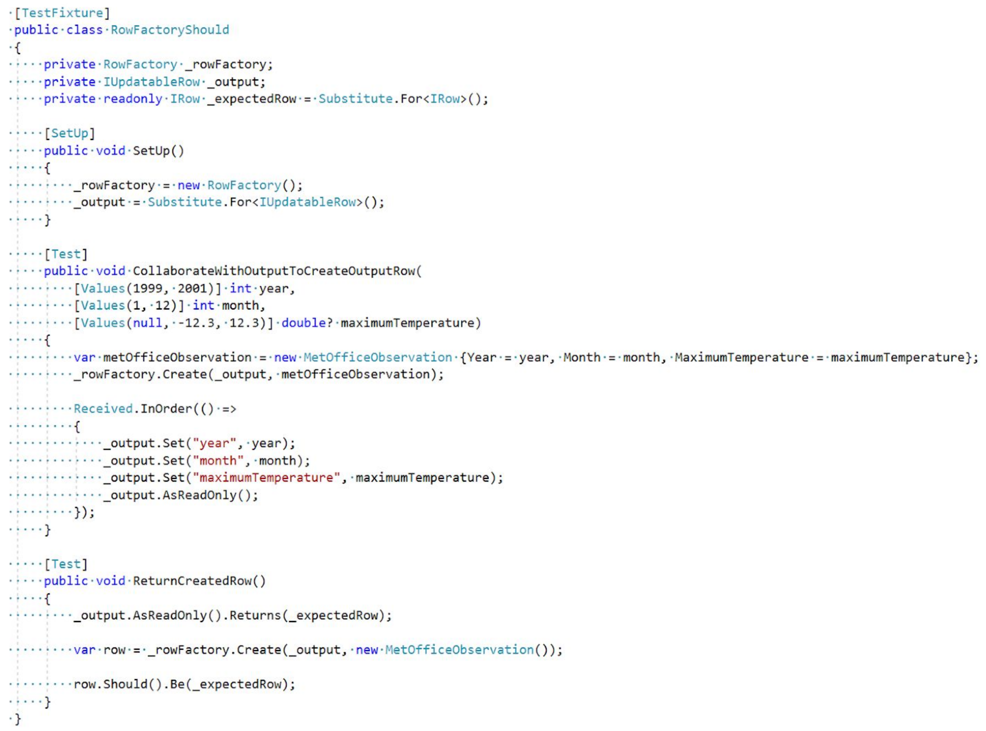

```
internal class InputReader
{
    public virtual IEnumerable<string> ReadLines(IUnstructuredReader input)
    {
        using (var streamReader = new StreamReader(input.BaseStream))
        {
            while (!streamReader.EndOfStream)
            {
                yield return streamReader.ReadLine();
            }
        }
    }
}
```



```
[TestFixture]
public class InputReaderShould
{
    private readonly IEnumerable<string> _lines = new[] { "Line1", "Line2", "Line3" };
    private InputReader _inputReader;
    private IUnstructuredReader _unstructuredReader;

    [SetUp]
    public void SetUp()
    {
        _unstructuredReader = Substitute.For<IUnstructuredReader>();

        _inputReader = new InputReader();
    }

    [Test]
    public void ReturnEachLineFromInputStream()
    {
        _unstructuredReader.BaseStream.Returns(StreamWithLines(_lines));

        var linesReturned = _inputReader.ReadLines(_unstructuredReader).ToList();

        linesReturned.Should().BeEquivalentTo(_lines);
    }

    private static MemoryStream StreamWithLines(IEnumerable<string> lines) => new MemoryStream(Encoding.UTF8.GetBytes(string.Join("\r\n", lines)));

}
```



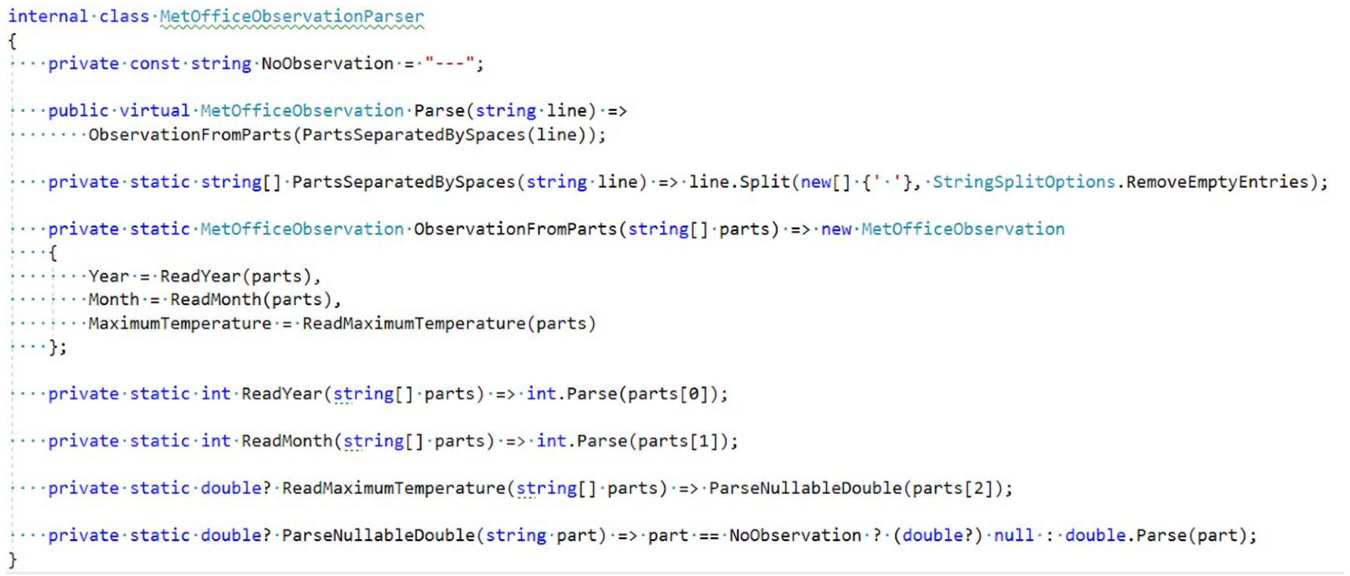
```csharp
 internal class RowFactory
 {
     public virtual IRow Create(IUpdatableRow output, MetOfficeObservation metOfficeObservation)
     {
         output.Set("year", metOfficeObservation.Year);
         output.Set("month", metOfficeObservation.Month);
         output.Set("maximumTemperature", metOfficeObservation.MaximumTemperature);
         return output.AsReadOnly();
     }
 }
```



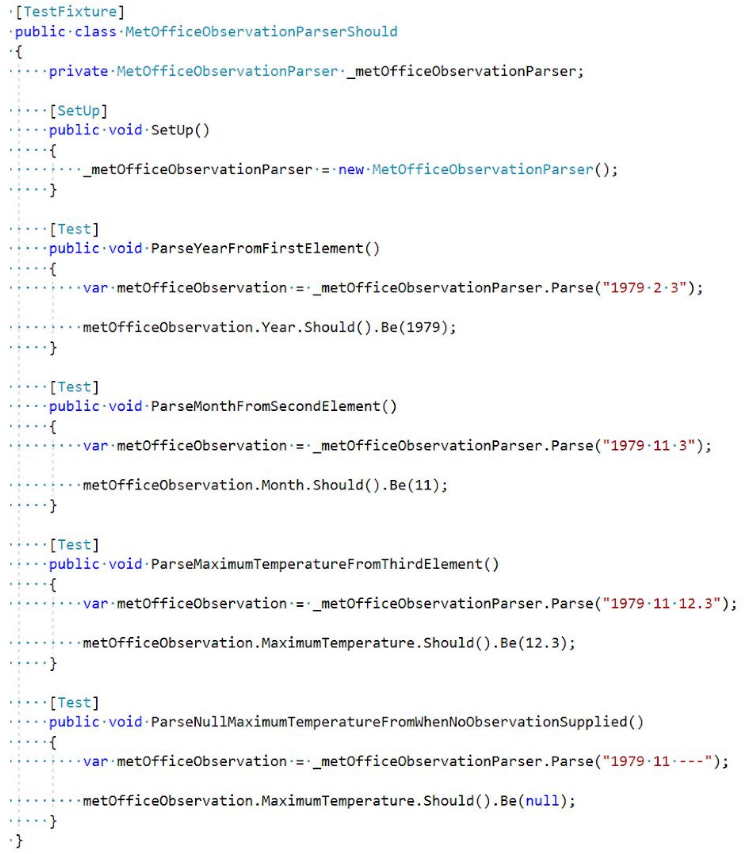
```
[TestFixture]
public class RowFactoryShould
{
    private RowFactory _rowFactory;
    private IUpdatableRow _output;
    private readonly IRow _expectedRow = Substitute.For<IRow>();

    [SetUp]
    public void SetUp()
    {
        _rowFactory = new RowFactory();
        _output = Substitute.For<IUpdatableRow>();
    }

    [Test]
    public void CollaborateWithOutputToCreateOutputRow(
        [Values(1999, 2001)] int year,
        [Values(1, 12)] int month,
        [Values(null, -12.3, 12.3)] double? maximumTemperature)
    {
        var metOfficeObservation = new MetOfficeObservation {Year = year, Month = month, MaximumTemperature = maximumTemperature};
        _rowFactory.Create(_output, metOfficeObservation);

        Received.InOrder(() =>
        {
            _output.Set("year", year);
            _output.Set("month", month);
            _output.Set("maximumTemperature", maximumTemperature);
            _output.AsReadOnly();
        });
    }

    [Test]
    public void ReturnCreatedRow()
    {
        _output.AsReadOnly().Returns(_expectedRow);

        var row = _rowFactory.Create(_output, new MetOfficeObservation());

        row.Should().Be(_expectedRow);
    }
}
```

Domain Tests

Familiar Territory



```
internal class MetOfficeObservationParser
{
    private const string NoObservation = "---";

    public virtual MetOfficeObservation Parse(string line) =>
        ObservationFromParts(PartsSeparatedBySpaces(line));

    private static string[] PartsSeparatedBySpaces(string line) => line.Split(new[] {' '}, StringSplitOptions.RemoveEmptyEntries);

    private static MetOfficeObservation ObservationFromParts(string[] parts) => new MetOfficeObservation
    {
        Year = ReadYear(parts),
        Month = ReadMonth(parts),
        MaximumTemperature = ReadMaximumTemperature(parts)
    };

    private static int ReadYear(string[] parts) => int.Parse(parts[0]);

    private static int ReadMonth(string[] parts) => int.Parse(parts[1]);

    private static double? ReadMaximumTemperature(string[] parts) => ParseNullableDouble(parts[2]);

    private static double? ParseNullableDouble(string part) => part == NoObservation ? (double?) null : double.Parse(part);
}
```



```csharp
[TestFixture]
public class MetOfficeObservationParserShould
{
    private MetOfficeObservationParser _metOfficeObservationParser;

    [SetUp]
    public void SetUp()
    {
        _metOfficeObservationParser = new MetOfficeObservationParser();
    }

    [Test]
    public void ParseYearFromFirstElement()
    {
        var metOfficeObservation = _metOfficeObservationParser.Parse("1979 2 3");

        metOfficeObservation.Year.Should().Be(1979);
    }

    [Test]
    public void ParseMonthFromSecondElement()
    {
        var metOfficeObservation = _metOfficeObservationParser.Parse("1979 11 3");

        metOfficeObservation.Month.Should().Be(11);
    }

    [Test]
    public void ParseMaximumTemperatureFromThirdElement()
    {
        var metOfficeObservation = _metOfficeObservationParser.Parse("1979 11 12.3");

        metOfficeObservation.MaximumTemperature.Should().Be(12.3);
    }

    [Test]
    public void ParseNullMaximumTemperatureFromWhenNoObservationSupplied()
    {
        var metOfficeObservation = _metOfficeObservationParser.Parse("1979 11 ---");

        metOfficeObservation.MaximumTemperature.Should().Be(null);
    }
}
```

You may still need to Mock the Clock!

What have we learnt?

Speed Run locally

Control Seams

Test doubles

Precision Small pieces

Clear purpose

Adapters

Patterns

Patterns

Patterns

Resources

Testing Patterns

github.com/bnathyuw/testing-patterns/wiki

Example implementation

github.com/bnathyuw/weather-data-iii

Dank je wel!