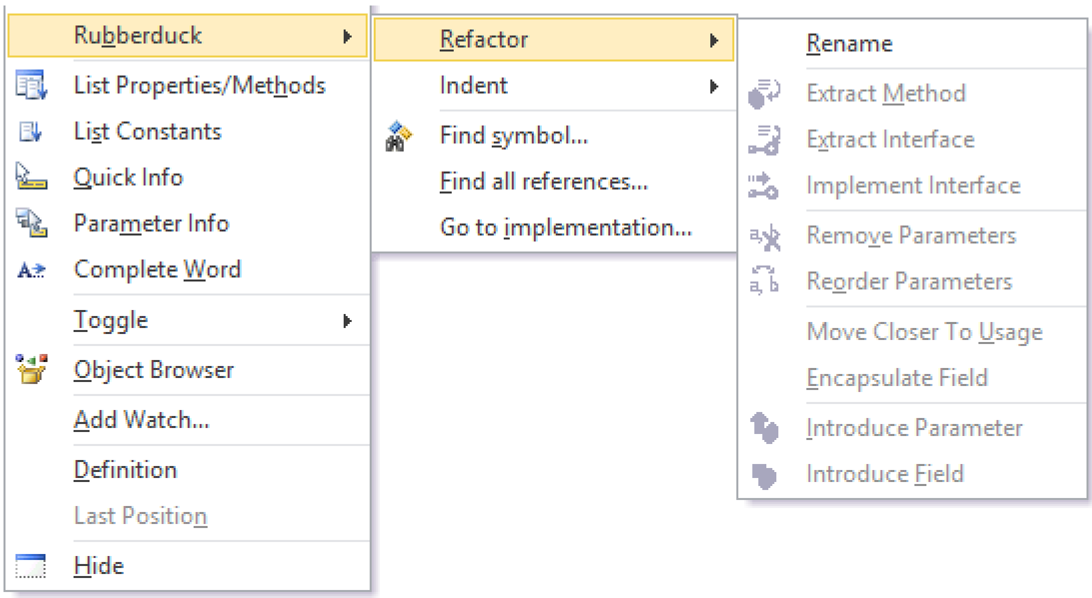


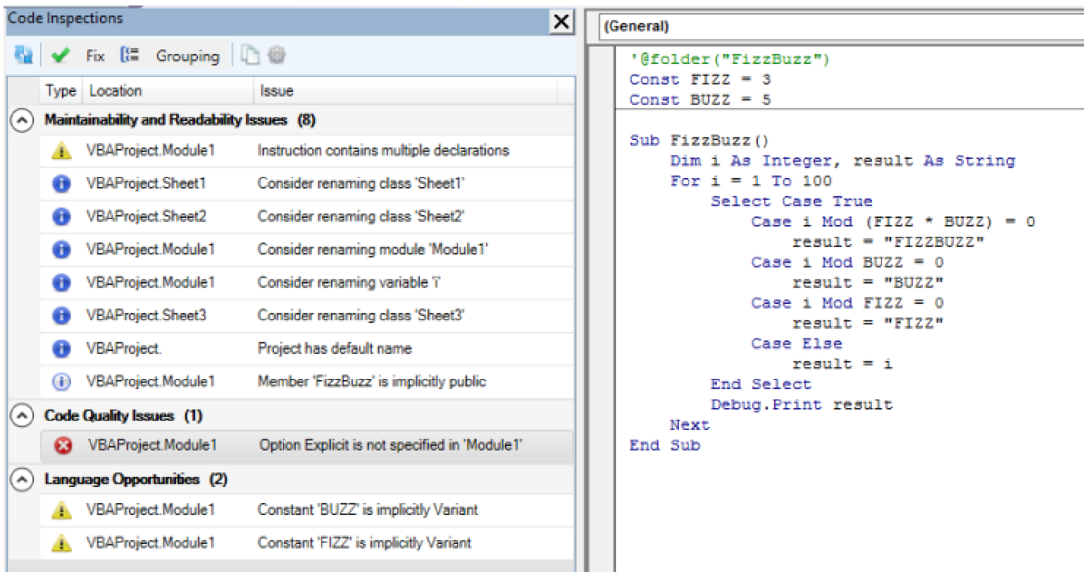
VBA Rubberducking (Part 4)

Posted on May 28, 2016May 28, 2016 by Rubberduck VBA
This post is the fourth in a series of post that walk you through the various features of the Rubberduck open-source VBE add-in.

- Part 1 (https://rubberduckvba.wordpress.com/2016/05/04/vba-rubberducking-part-1/) introduced the **navigation** features.
- Part 2 (https://rubberduckvba.wordpress.com/2016/05/14/vba-rubberducking-part-2/) covered the **code inspections**.
- Part 3 (https://rubberduckvba.wordpress.com/2016/05/18/vba-rubberducking-part-3/) featured the **unit testing** feature.

Refactorings

At first we were happy to just be able to *inspect* the code.


Code Inspections

Fix | Grouping

Type	Location	Issue
Maintainability and Readability Issues (8)		
⚠	VBAProject.Module1	Instruction contains multiple declarations
ℹ	VBAProject.Sheet1	Consider renaming class 'Sheet1'
ℹ	VBAProject.Sheet2	Consider renaming class 'Sheet2'
ℹ	VBAProject.Module1	Consider renaming module 'Module1'
ℹ	VBAProject.Module1	Consider renaming variable 'i'
ℹ	VBAProject.Sheet3	Consider renaming class 'Sheet3'
ℹ	VBAProject.	Project has default name
ℹ	VBAProject.Module1	Member 'FizzBuzz' is implicitly public
Code Quality Issues (1)		
✖	VBAProject.Module1	Option Explicit is not specified in 'Module1'
Language Opportunities (2)		
⚠	VBAProject.Module1	Constant 'BUZZ' is implicitly Variant
⚠	VBAProject.Module1	Constant 'FIZZ' is implicitly Variant

(General)

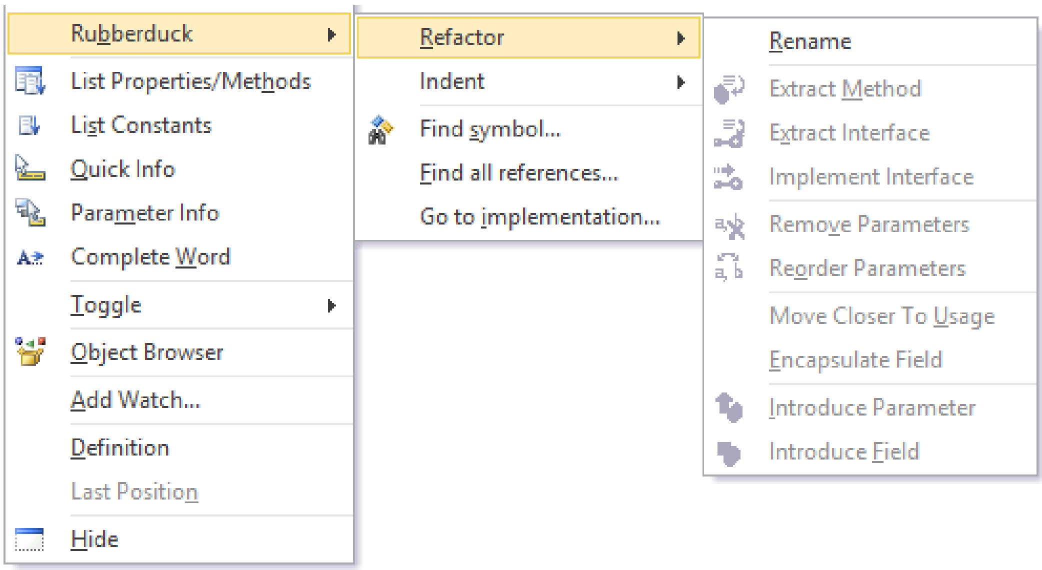
```
'@folder("FizzBuzz")
Const FIZZ = 3
Const BUZZ = 5

Sub FizzBuzz()
    Dim i As Integer, result As String
    For i = 1 To 100
        Select Case True
            Case i Mod (FIZZ * BUZZ) = 0
                result = "FIZZBUZZ"
            Case i Mod BUZZ = 0
                result = "BUZZ"
            Case i Mod FIZZ = 0
                result = "FIZZ"
            Case Else
                result = i
        End Select
        Debug.Print result
    Next
End Sub
```

Quickly we realized "inspection quick-fixes" could be something else; some of the inspections' quick-fixes are full-fledged automated *refactoring* operations. Renaming an identifier – *and doing it right* – is very different than just Ctrl+H/replace an identifier. Manually removing an uneeded parameter in an existing method breaks all call sites and the code no longer even compiles; Rubberduck sees all call sites, and knows which argument to remove everywhere to keep the code compiling.. and it's much faster than doing it by hand!

Rubberduck 1.3 had *Rename* and *Extract Method* refactorings; v1.4.3 also had *Remove Parameters* and *Reorder Parameters* refactorings.

Rubberduck 2.0 introduces a *few* more.



The context menu commands are enabled depending on context; be it the current *parser state*, or the current selection.

Rename

That's a pretty well-named refactoring. It deals with the impacts on the rest of the code base, of renaming pretty much any identifier.

Extract Method

Pretty much completely rewritten, v2.0 *Extract Method* refactoring is becoming pretty solid. Make a valid selection, and take that selection into its own member, replacing it with a call to the extracted code, all parameters and locals figured out for you.

Extract Interface

VBA supports interface inheritance; Rubberduck makes it easy to pull all public members of a module into a class that the original module then **Implements**. This is VBA's own way of *coding against abstractions*. Unit tests love testing code that's *depending on abstractions, not concrete implementations*, because then the tests can provide ("inject") *fake* dependencies and test the applicative logic without triggering any unwanted side-effects, like displaying a message box, writing to a file, or to a database.

Implement Interface

Implementing all members of an interface (and *all members* of an interface *must* be implemented) can be tedious; Rubberduck automatically creates a stub method for every member of the interface specified in an **Implements** statement.

Remove/Reorder Parameters

Reworking a member's signature is always annoying, because then you have to cycle through every single call site and update the argument list; Rubberduck knows where every call site is, and updates all call sites for you.

Move Closer to Usage

Variables should have the smallest possible scope. The "scope too wide" inspection uses this refactoring to move a declaration just above its first usage; it also works to rearrange "walls of declarations" at the top of a huge method you're trying to cut into more manageable pieces.

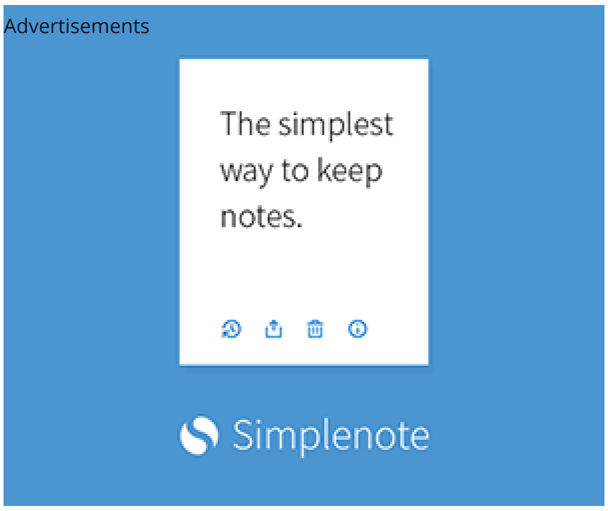
Encapsulate Field

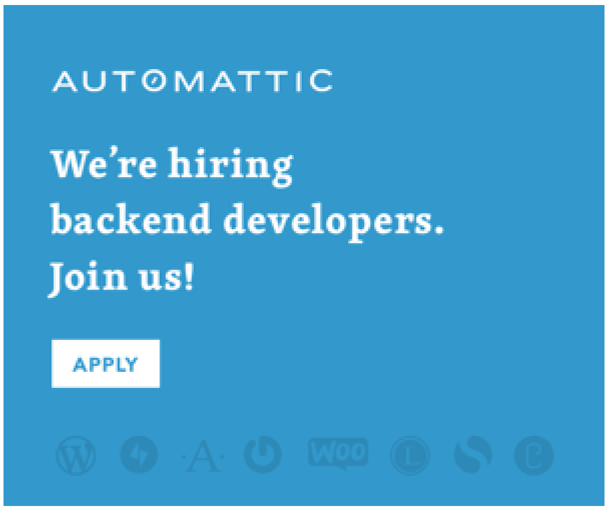
Fields are internal data, implementation details; objects shouldn't expose public fields, but rather, *encapsulate* them and expose them as properties. Rubberduck turns a field into a property with only as much effort as it takes to name the new property.

Introduce Parameter/Field

Pretty much the antagonist of *move closer to usage*, this refactoring promotes a local variable to a parameter or a field, or a parameter to a field; if a new parameter is created, call sites will be updated with a "TODO" bogus argument that leaves the code uncompilable until an argument is supplied for the new parameter at all call sites.

More refactorings are planned for 2.1 and future versions, including **Inline Method** (the inverse of *Extract Method*), to move the body of a small procedure or function into all its call sites. Ideas for more refactorings and inspections? Suggest a feature (https://github.com/rubberduck-vba/Rubberduck/issues/new)!





Posted in OOP, open-source, rubberduck, vbaTagged add-in, code analysis, code-inspections, encapsulate field, extract interface, extract method, find all references, identifier resolution, implement interface, introduce field, introduce parameter, move closer to usage, oop, parsing, refactoring, remove parameters, rename, reorder parameters, rubberduck, unit-testing, v2.0, vba, vba tools, vbe

Published by Rubberduck VBA

I'm Mathieu Guindon (Microsoft MVP Office Apps & Services, 2018), you may have known me as "Mat's Mug" on Stack Overflow and Code Review Stack Exchange. I manage the Rubberduck open-source project, whose goal is to bring the Visual Basic Editor (VBE) - VBA's IDE - into the 21st century, by providing features modern IDE's provide. *View all posts by Rubberduck VBA*