

OOP Design Patterns: The Builder

Posted on ~~July 3, 2018~~ ~~July 3, 2018~~ by Rubberduck VBA

The *Builder Pattern* is rarely something you *need*. Often a *Factory Method* does the job just fine, as far as *creating object instances* goes. But sometimes, creating an object in a valid state would require a *Create* method with many parameters, and that gets annoying.

There's something rather elegant about chained member calls that build an object. The methods of a *FooBuilder* class return *Me*, so the calling code can chain the member calls and build the object in a single, expressive statement:

```
1  Set pizza = builder _  
2      .OfSize(Medium) _  
3      .CrustType = Classic _  
4      .WithPepperoni _  
5      .WithCheese(Mozza) _  
6      .WithPeppers _  
7      .WithMushrooms _  
8      .Build
```

The *Build* method returns the *product*, i.e. the resulting object.

So a basic (and rather flawed) builder class might look like this:

```
1  Private result As Pizza  
2  
3  Private Sub Class_Initialize()  
4  Set result = New Pizza  
5  End Sub  
6  
7  Public Function OfSize(ByVal sz As PizzaSize) As PizzaBuilder  
8  If result.Size = Unspecified Then  
9  result.Size = sz  
10 Else  
11 Err.Raise 5, TypeName(Me), "Size was already specified"  
12 End If  
13 Set OfSize = Me  
14 End Function  
15  
16 Public Function WithPepperoni() As PizzaBuilder  
17 result.Toppings.Add(Pepperoni)  
18 Set WithPepperoni = Me  
19 End Function  
20  
21 '...  
22  
23 Public Function Build() As IPizza  
24 Set Build = result  
25 End Function
```

Every “builder method” is a Function that returns Me, and may or may not include a bit of logic to keep the result valid. Then the Build function returns the encapsulated and incrementally initialized result object.

If the return type of the Build function is an interface (that the result object implements), then the calling code can treat all pizzas equally (assuming, say, ClassicCrustPizza, PanPizza, ThinCrustPizza are different acceptable implementations of the IPizza interface... this is where the pizza example really crumbles), and the interface can very well *not* expose any Property Let members.

Considerations

The builder pattern is fun and *very good to know*, but it’s very rarely something that’s *needed*. But for these times when you *do* need it, there are a number of things to keep in mind:

- **No temporal coupling:** the order in which the calling code calls the builder methods should make no difference.
- **Builder methods may not be invoked:** if a pizza without a Size isn’t a valid Pizza object, then there shouldn’t be a builder method for it; either provide sensible defaults, or make a parameterized factory that creates the builder with all the non-optional values initialized.
- **Repeated invocations:** the calling code might, intentionally or not, invoke a builder method more than once. This should be handled gracefully.
- **Readability:** if the *fluent API* of a builder isn’t making the code any easier to read, then it’s probably not worth it.

You’ll think of using a builder pattern when a factory method starts having so many parameters that the call sites are getting hard to follow: a builder *can* make these call sites easier to read/digest.

This SoftwareEngineering.SE answer

(<https://softwareengineering.stackexchange.com/a/345704/68834>) describes the actual GoF Builder Pattern (see Design Patterns: Elements of Reusable Object-Oriented Software

(<http://rads.stackoverflow.com/amzn/click/0201633612>)), which takes it a notch further and makes the builder itself abstract, using a much better example than pizza. I warmly encourage you to read it; even though the code isn’t VBA, the principles are the very same regardless.

ADVERTISEMENT

Advertisements

AUTOMATTIC

```
<?php find_developers( [  
    'language' => PHP,  
    'specialty' => SCALING,  
    'location' => ANYWHERE,  
] );
```

APPLY



REPORT THIS AD

AUTOMATTIC

**You don't need to go to
an office to write code.
Work with us!**

APPLY



REPORT THIS AD

Posted in [OOP](#), [tutorials](#), [vba](#) Tagged [builder](#), [design-patterns](#), [oop](#), [tutorial](#), [vba](#)



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](#)



	-26%	-13%		-30%	-25%	-28%
	R\$ 169,99	R\$ 259,90	R\$ 513,00	R\$ 159,99	R\$ 89,90	R\$ 579,99

REPORT THIS AD