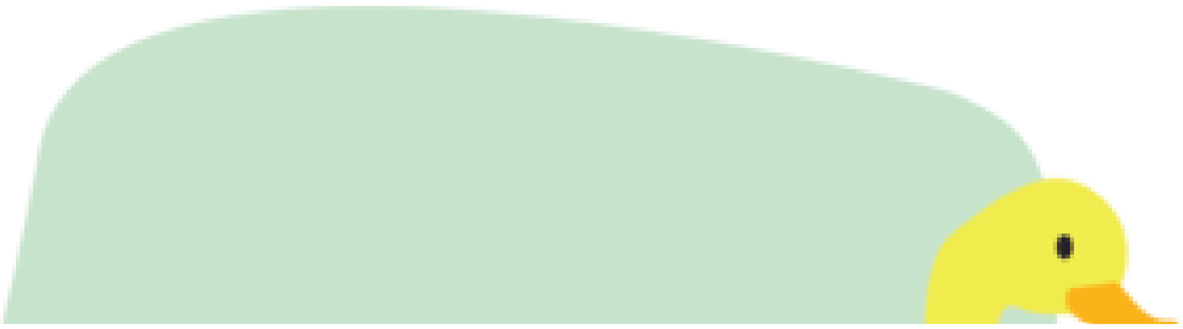
DISQUS Home 9+ Inbox Discover Discuss

Community

256 yegor256.com

90 Comments • Created 6 months ago



Seven Virtues of a Good Object

Martin Fowler says: A library is essentially a set of functions that you can call, these days usually organized into classes. Functions organized into classes? With all due respect, this is wrong. And it is a very common misconception of a class…

(yegor256.com)

90 Comments

Recommend 2 Share Sort by Newest

Join the discussion…

Loup Vaillant • a month ago

Can't resist posting, I'm too… horrified. I agree with too few of your points. First, the good:

He is immutable. Yes! Though I think you're not going far enough: if an object doesn't display a referentially transparent interface (same call => same results), I don't care if it's mutable or not: it let side effects leaks through it, and that's not ideal. Effects should be isolated.
His name is not a job title… sounds reasonable.
His class is either final or abstract… Well, inheritance is bad anyway, so, yes of course.

Now the bad:

He exists in real life… I see no justification.
He is unique… I see no justification.
Why do you believe those two matter at all? If you can't pinpoint a reason, maybe you should reassess those assumptions

Now the ugly:

He works by contract… Bad example. Why depend on a whole freaking interface when you could just take the data itself? Why don't your class just take a `byte[]` as an input? It will make everything easier: no need for that pesky interface at all (less code), do dependence upon that interface, just `byte[]` (less coupling), any `byte[]` producer can work with your objects (more flexibility)… Your example doesn't make sense.

His class doesn't have anything static… Bad justification. Functional Programming is build on static methods, and is very good at composition. Since you like immutability, you can do it too. Static methods don't have to be thread unsafe: if they are referentially transparent (and they can be), then they are thread safe… by definition. And you don't need to mock those. Just test them in isolation before you use them in other tests. Like the FP folks do.

The only thing left for your hate of static methods is that they don't look OOP. But that alone is not a valid justification for anything.

1 • Reply • Share ›

David Raab ➦ Loup Vaillant • a month ago

> He works by contract… Bad example. Why depend on a whole freaking interface when you could just take the data itself? Why don't your class just take a `byte[]` as an input? It will make everything easier: no need for that pesky interface at all (less code), do dependence upon that interface, just `byte[]` (less coupling), any `byte[]` producer can work with your objects (more flexibility)… Your example doesn't make sense.

I want to add something to it. Every time someone uses an interface with a single method definition, or creates a class with 2 methods (constructor + method) you already knew something goes wrong. Those cases are always better represented by static methods and wrapping it in classes and adding interfaces add nothings to it, just boilerplate.

• Reply • Share ›

Marcos Douglas Santos ➦ David Raab • a month ago

There are nothing wrong if you have an interface with a single method definition. Maybe a class with only one method is strange, but not interface.

• Reply • Share ›

David Raab ➦ Marcos Douglas Santos • a month ago

An interface with a single method defined makes no sense because that is already what you get by a normal static method definition. When you define an interface with a single method the only thing you can do is to call this only method. You can't do anything else with such an interface. The only thing that you then enforce are the arguments of this method and what it returns. And you already get that with a single static method.

An interface like Yegor did in other post like

```
interface Text {
    string Method(string arg)
}
```

is identical to a static function

```
string Method(string arg)
```

you don't get any more benefit from the fact that you created an interface and wrap it in lots of classes. You can implement a lot of classes with that interfaces. The same as you can implement a lot of static methods that get and returns a string.

• Reply • Share ›

Marcos Douglas Santos David Raab • a month ago

You are missing the point about interfaces.
Example: you can have a class with 10 methods but some parts of your system you only need an object that do something. Imagine an interface like:

```
interface Task {
  void run();
}
```

You have an instance of Foo class and this class implements Task... do you understood me? You do not need a hierarchy of classe that have run method, you only need an object that implements this method. The object could do more things, but in that moment you only need to call run method.

• Reply • Share ›

David Raab Marcos Douglas Santos • a month ago

I don't miss the point about interfaces, i think you underestimate functions or static methods ;) What you described is by the way just a callback. Your passing an Object that implements the Task interface and you later can call "run" on it that do something. Without an interface and just static method, in C# you just write the following code

```
static SomeMethod(Action task) {
    ....
    // Execute the provided task by the user, and do whatever Tas
    task();
    ....
}
```

And here you have it. The user now can pass a static method as a

And here you have it. The user now can pass a static method as a parameter or create an anymous function (lambda) on the fly. In your version you also can provide a full object that implements the Task interface, that is what you described, but it has no benefits. It is just more complicated and you have to write more code. While the SomeMethod function in your version would look nearly identical...

```
static SomeMethod(ITask task) {
    ....
    // Execute the provided task by the user, and do whatever Tas
    task.run();
    ....
}
```

... it gets a lot more complicated to actiually provde code that gets executed. In my version you just can create a lambda. For example to Print a line, you write the following. Just a single line of code.

```
SomeMethod(() => Console.WriteLine("Task gets executed"));
```

In your version you have to write the following stuff, to execute your SomeMethod() function with your interface.

```
public class PrintLine : Task {
    // Empty Constructor
    public PrintLine() {}
    // Our Run Method
    public void Run() {
        Console.WriteLine("Task gets executed");
    }
}

// We also need an useless object, that doesn't even make sense!
var task = new PrintLine();

// pass the object as a callback
SomeMethod(task);
```

You need to create a class, implement the interface. And then instantiate it just to call the "Run" method on it. And the point is, it is also not more OO. The only purpose of the Task interface is to hold a single method that gets called. You absolutely can't do anything else with that object. If the purpose of an class is to only execute a single method, then you can just write a single static method. You always can convert the pattern

```
class Foo {
    public Constructor(T1 arg1, T2 arg2)
    public T3 Run()
    public T3 Run()
}
```

just to

```
T3 Foo(T1 arg1, T2 arg2);
```

An interface with a single method is always replaceable by a static method. Wrapping it in an interface has absolutely no benefit at all. Just think about it. If the only purpose is to just call the only method available on it, then you just have a static method.

It is also by the way possible to pass a static function directly as a parameter. For example something like this.

```
public static PrintLine() {
    Console.WriteLine("Task gets executed");
}
SomeMethod(PrintLine);
```

Note that when you just write "PrintLine" and not "PrintLine()" then the function doesn't get executed, the function gets passed in as a parameter, and SomeMethod can execute the function whenever it wants. And you always have full control what SomeMethod should execute. This is just normal higher-order functions from functional programing. C# have all of these features, but like i said.

Java should also have all of these features.

Or just watch the following (Starting at 2:09):

• Reply • Share ›

Marcos Douglas Santos ➦ David Raab • a month ago

> I don't miss the point about interfaces, i think you underestimate functions or static methods ;) What you described is by the way just a callback.

I'm an Object Pascal programmer (don't confused Delphi programmer, but I use Delphi too). Pointers, functions, create and destroy objects "by hand", this is my world.

The Object Pascal have callback too, but we call events -- by the way, C# is a copy from Delphi language + Java... the same author that came out the Borland, by the way. So, I know procedural programming. Events or callbacks. Pointers. But I don't think this is the best way to do.

You used a function in your example. Ok. But our example is so simple. If we need to execute more than one function? You will pass a record (Pascal) / struct (C) with pointers to another functions? Better to use a class, don't?

> In my version you just can create a lambda.

Please, do not compare functional vs OOP. If you write more or less isn't an argument.

Finally, the video is about Python and it suggests do not create classes... Python not is a true object-oriented language so this video doesn't count, sorry! :)

• Reply • Share ›

David Raab ➦ Marcos Douglas Santos • a month ago

> Pointers, functions, create and destroy objects "by hand", this is my world.

The way how you use functions still differs a lot compared to just a procedural language with pointers. And you usually don't create anything by hand. You absolutely need a high-level automatic memory management, if you don't have that, then Closures are not really possible. And this is a fundamental concept of Functional Programming.

> The Object Pascal have callback too, but we call events

A callback and an event are two different things. The idea of a callback is more to directly use the provided function and also evaluate the return value of a function. While an event exists more that other (multiple) things can register to an event and get a notification. At least in C# you have both (Delegate and Events) and it makes sense to differ both.

> You used a function in your example. Ok. But our example is so simple. If we need to execute more than one function?

Then you provide multiple arguments.

> You will pass a record (Pascal) / struct (C) with pointers to another functions? Better to use a class, don't?

You never do such things in a functional language, that would be ugly.

> Please, do not compare functional vs OOP. If you write more or less isn't an argument.

Languages like Java, C# and all of the other languages are not restricted to OO only and have a lot of functional programing in it. It would be stupid to not use these features. And the discussion started that Yegor thought a static method would not be functional, what isn't the case.

> Finally, the video is about Python and it suggests do not create classes... Python not is a true object-oriented language so this video doesn't count, sorry! :)

Whether it is Python or not is completely irrelevant. He shows a concept that a class with just an constructor and a single-method is stupid and isn't really a real class. This is a higher concept that doesn't change, wether you use Python, Java, C# or any other OO language that currently exists or will be created in the future.

• Reply • Share ›

Marcos Douglas Santos David Raab • a month ago

Delegates is a pointer to function. Event is a layer to use delegate more cool. Both could be implements in any procedure/functional language, like Pascal.
The differences about Object Pascal and C# (or others) if a callback is a "delegate", "event" or a callback... well, better we forget it.

> (...)discussion started that Yegor thought a static method would not be functional, what isn't the case.

In that case, lets wait the Yegor's answer.

> He (the video) shows a concept that a class with just an constructor and a single-method is stupid and isn't really a real class

But I said that a class, not an interface, with only one method is strange. So I agree with you that something could be wrong... maybe... but if an interface have only one method, I don't see how this is could be stupid.

• Reply • Share ›

David Raab Marcos Douglas Santos • a month ago

> Delegates is a pointer to function. Event is a layer to use delegate more cool. Both could be implements in any procedure/functional language, like Pascal.

That is true, but the difference is still important. I mean, lets look at some OO words. You use the "Decorator Pattern"? What does the Decorator Pattern provide besides Classes and Methods? Well, absolutely nothing. What is MVC? Well also just Classes and method. When you use immutable objects, does that change? Nope, still just classes and methods. Does it change with "Dependency Injection", "Factory Methods", "Singleton", "Adapter Pattern", "Facade Pattern", "Proxy Pattern", "Iterator", "Observer", ...

Well no, all of them just uses classes and methods. Why a lot of different names for them? Because all of them describe a specific technique. And you have to learn this technique and what they do. And with "callback" and "event" it is the same. Yes, they are both function pointers, well event is usually a list of function pointers while a callback is a single function. But they both describe a completely different technique and how you use them. So it is important to differentiate both.

And that is by the way why i'm saying that programing in a functional language differs a lot, and just knowing functions isn't already "functional". Also in a functional world you have your patterns to learn. From Currying, Monads, Monoids, Immutability and so on. Even if all of them just uses functions under the hood.

http://www.ndcvideos.com/#/app...

> But I said that a class, not an interface, with only one method is strange. So I agree with you that something could be wrong... maybe... but if an interface have only one method, I don't see how this is could be stupid.

Well in that case the difference doesn't matter so much. When you have a class with a single method you only can call this single method. If you have an interface with a single method defined, then you also can just call this one single method on the concrete object whatever the real implementation is.

But okay "stupid" sound a little bit aggressive. But what i meant was that it doesn't make much sense. If your implementation only has a single method, and the only thing you can do is to call that method, and you can't do anything else with it, then you just can use a function. Well it would be different if the class would be mutable and every time you call that method it changes it mutable state. But i think we agree both that classes should be immutable. And also Yegor uses Immutability. That is also the point why you just can swap it with a single static method/function.

You don't mutate anything. Everything you pass to a constructor you also can pass to a static method.

1 • Reply • Share ›

Marcos Douglas Santos David Raab • a month ago

Even using different thought, I think we agree on many things.
OK, you've convinced me that we can use static methods, that is not (much) different to use a constructor. But I still think is better not use them for a "best design". But I understand your point and, once again, thanks for the links and videos.

• Reply • Share ›

David Raab Marcos Douglas Santos • a month ago

Yes, my intention also was not to say it is best practices. My intention is the following.

If you define classes or an interface with a single method, then your design is not OO. There are two things someone should do here, if that happens.

1) Re-look at the design and try to make it OO.

2) If you don't see a chance for the first or how to fix it, or you think that your class/interface with a single method have to exists, rewrite it as a static method. Because what you really have here is really just a "Utility method" and the fact you are wrapping it in its own class doesn't change this fact.

So the point is, if you have Utility methods write a Utility Class! There is nothing wrong with this. Your Utility methods doesn't get "not utility" if you wrap every single utility method in its own class. And i think that is important. Why?

A lot less code. Less code is easier to read, thus easier to understand. It makes it clear which things are just a Utility methods and what is a real class. You also have to write less code. That is also important because it embraces to do the right thing.

A Utility method should not be part of the main interface. But if you have to write a full class in its own file, creating an interface to provide your utility class, then chances are high someone just add the utility method directly to the class or to the public interface and just do the wrong thing.

It should always be that "doing the right thing" is easy. If you have a design where "doing the right thing" is hard, there is just a design-flaw somewhere.

• Reply • Share ›

Yegor Bugayenko author ↗ Loup Vaillant • a month ago

This article discusses static methods and functional programming, you may find it interesting too: http://www.yegor256.com/2015/0...

• Reply • Share ›

abhi • 2 months ago

what is your take on the book Effective Java? should an OO purist learn from it and follow it religiously ?

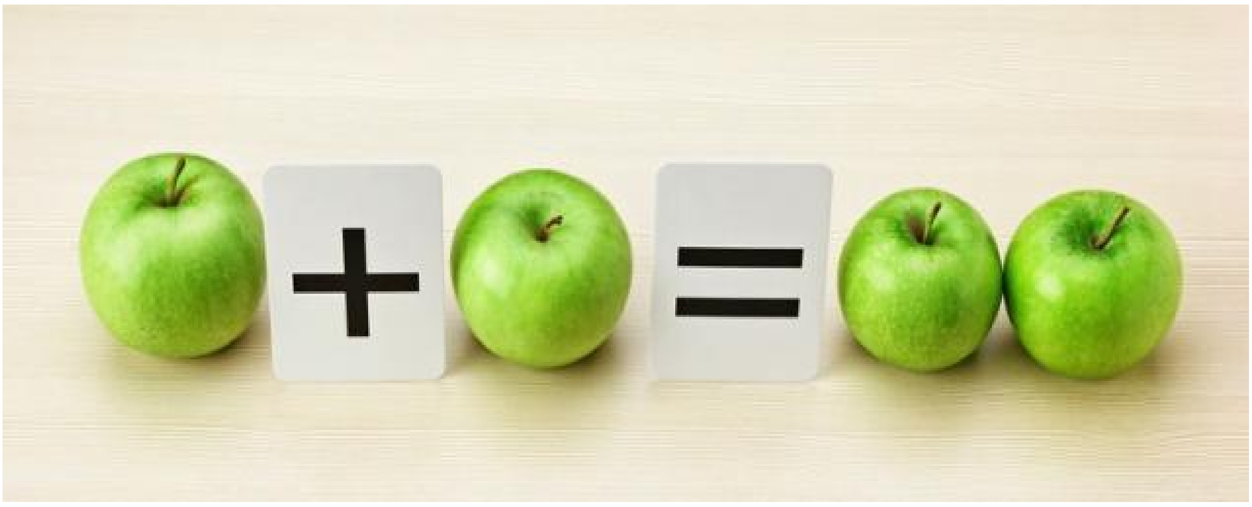
• Reply • Share ›

Yegor Bugayenko author ↗ abhi • 2 months ago

It's a bad guidance for bad programmers. I would suggest to stay away from this book.

• Reply • Share ›

Zack • 2 months ago



Objects can be aggregated. The idea that you can't sum objects with "+' or multiply objects with "*" is crazy. If you have 1 apple and you add (+) 1 apple then you have 2 apples. Our simulation improves when we look at objects as real things that can be aggregated in a similar fashion to real things.

1) It makes more sense than substituting methods for operations that define creational actions (i.e., add, subtract, multiply, divide).

2) It is easier to write and read.

//example using object thinking code

ListOfApples apples = (apple1 + apple2 + apple3 + apple4 + apple5 + apple6);
apples = (apples + apple7 + apple8 + apple9 + apple10);

if((apple1 + apple2 + apple3).Weight() == (apple7 + apple8 + apple9).Weight()){
Console.WriteLine("These groups of apples weight the same, what are the odds?");
}

//example using "proceduralized" object thinking code

ListOfApples apples = new ListOfApples(apple1, apple2, apple3, apple4, apple5, apple6);
apples = new ListOfApples (apples, new ListOfApples(apples, apple7, apple8, apple9, apple10));

if((new ListOfApples(apple1 + apple2 + apple3).Weight()) == (new ListOFApples(apple7 + apple8 + appl9).Weight())){
Console.WriteLine("These groups of apples weight the same, what are the odds?");
}

//Some other interesting things we could do with object thinking

Pound firstWeight = new Pound(new Number(5)); //pound is a unit so the incremental increase is 1.
Pound secondWeight = firstWeight++; //magnitude 6
Pound thirdWeight = (secondWeight * (new Number(10))); //magnitude 60

//..................student1 + student2 => listOfStudents, ListOfReportCards takes listOfStudents
Printer((new PrintJob (new ListOfReportCards(student(new Name ("John Doe"), new Grade(3.0)) +
student(new Name ("Sally May"), new Grade(2.0)) +
student(new Name ("Freddy Blue"), (new Grade(4.0)) +
)
) * (new Number(3))
)
).Print();

• Reply • Share ›

David Raab ➦ Zack • a month ago

At first, summing/aggregating something is something different as you show with your image or show with your code. Adding something means that you have two things and those two things get combined into a single new item. What you provide is just creating Lists what is absolutely not what a "sum" does.

If you do "5 + 5" the result you get is also 10. You don't get a List that contains two numbers and both numbers are 5. And that is what you actually describe with your image. A "real" sum operation on two apples would be a new apple that has the size/weight of the two apples combined. Not just two single apples. And with this idea behind, a sum on apples doesn't exists. You can't just take two apples and create one bigger apple. You can take two apples, but that is not adding two apples together.

To your code examples. Both are procedural. Procedural programing means:

1) Asking for data
2) Evaluating the data
3) Based on the evaluation decide what to do

```
if( (apple1 + apple2 + apple3).Weight() == (apple7 + apple8 + apple9).Weight()
    Console.WriteLine("These groups of apples weight the same, what are the odd
}
```

In your first example above you create two lists. On both your Lists you ask for the data "Weight". Compare those data and then decide if the Weight matches what to do. And in your second example, nothing changes, to that idea.

```
if( (new ListOfApples(apple1 + apple2 + apple3).Weight()) == (new ListOFApples(
    Console.WriteLine("These groups of apples weight the same, what are the odd
}
```

Also in this example you still create two list, ask for the Weight of the List, compare it, and when it matches you do something. Both code are procedural. None of your provided code is more OO. The fact that the List is created through a Sum operation, or that you create the List manual doesn't change the fact that it is still procedural. In order to be more "OO", you always need to consider "Tell, don't ask". https://pragprog.com/articles/...

• Reply • Share ›

Yegor Bugayenko author → Zack • 2 months ago

Zack, would be great if you can format Java code in all of your answers, using this mechanism: https://help.disqus.com/custom... Would be much easier to read

• Reply • Share ›

zack → Yegor Bugayenko • 2 months ago

Will do.

• Reply • Share ›

Zack • 2 months ago

It seems to me that many of the things developers are discussing with relation to objects have nothing to do with objects. But they have a lot to do with programming. A good example is the "exception object". Have you ever heard of such a thing in the real world? Is it physical thing or a concept? Nope. It's a programming technique for dealing with something that should never even happen in an object simulation. The only time it should (maybe) happen is when there is a problem outside the program (i.e., can't read file because of corruption). If you are confused it is because you have the wrong paradigm in your head.

Let me explain:

1) Programming is the design and production of executable instructions that control a computer. There are stages: logical programming, coding, and compiling. The ultimate goal is to control the computer. I will say this again: you are TRYING to CONTROL the computer. It doesn't matter why, what language you use, what machine code is used, what hardware is used. The idea is the same. Programming is a machine technology.

2) A simulation is a model that behaves like the system it represents (as opposed to an inanimate look-alike). Lots of simulations are analog. For example, you can simulate liquefaction during an earthquake and its effect on a building as shown in this video:

. Computer simulation is the design and production of simulations that operate on computers. It is a problem solving approach (simulating) that uses a machine technology (computers).

3) Object Simulation (otherwise called OOD/OOP) is the design and production of a computer simulation that behaves like an OBJECT system. Pay attention here. Usually the purpose of a simulation, such as in science and engineering, is to study the system being modeled. However, the purpose of an object simulation is to use the simulation as a proxy for the modeled system, because that only exists in mental space. Let me reiterate: you are TRYING to make a simulation that you can use as a proxy for a useful but imaginary system. In an object simulation the model is composed of entities known as objects who are intelligent anthropomorphisms of real world things (including concepts like time and distance). Notice how this is different from a regular simulation based primarily on actual, dumb/dead things. Objects are imaginary, animated analogs of their real-world counterparts that possess intelligent abilities. This is necessary because problem solving with things that don't do anything is very hard when your method of problem solving is

MODELING behavior. What the user sees as "the program" when they click the executable is actually "the simulation" implemented with a program. We use computers to simulate object systems that only exist in our minds. Object simulation is a problem solving approach that employs a machine technology.

Get it?

1 • Reply • Share ›

尤川豪 ➦ Zack • 2 months ago

Nice explanation. Very inspiring. Thanks.

• Reply • Share ›

Zack ➦ 尤川豪 • 2 months ago

Thanks. Writing helps me clarify my understanding.

1 • Reply • Share ›

Zack • 2 months ago

I am learning software architecture in C# (at the present) with the aim of becoming a professional. Your website has been a tremendous help! Thank you!

As I learn Object Thinking I have been working to un-brainwash myself from the procedural programming I learned in college. Reading these articles over the last few days made me realize that object thinking is efficient enough to work - and I mean all the way down to primitives. I didn't understand this before because proceduralists have a mantra that objects use too much memory to be the basic unit of work. They point out that instantiation takes "too long" and the heap space is "not enough" to hold all of the needed objects (laugh), and that algorithms can be performed much faster anyway with functions. This appears everywhere, I mean EVERYWHERE, even in so called OOP books (e.g. they use literal primitives to "save space and time"). I simply assumed they were right.

So, when you and others such as David West proposed a full-scale object orientation, I wasn't convinced that it would be feasible. However, I liked the idea very much. To investigate whether an object paradigm works fast enough to solve problems I conducted a simple experiment. My results demonstrate that object problem solving is more than efficient enough although much slower than procedural methods.

The Experiment: Find the total area of a bunch of randomly sized squares. If you want the actual code you can ask Yegor for my email and I will send it to you.

```
Class TestObjectiveVsProceduralOperationTime
{
//compute the total area of random squares with side: 1 <= length <= 10
TimeSpan stopTime;
TimeSpan startTime;
TimeSpan TotalTimeOverRuns = new TimeSpan();
int totalArea = 0;
//int iterations = 1000000;
//int iterations = 100000;
```

```
int iterations = 10000;
//int iterations = 1000;

Console.WriteLine("");
Console.WriteLine("Run 100 times: Iterate " + iterations.ToString() + " times and compute
the total area of random squares with side 1 <= length <= 10");

//procedural implementation: run test 100 times and compute average time
Console.WriteLine("");
Console.WriteLine("Execute procedural implementation:");

int side;
Random random = new Random();
for (int runs = 0; runs < 100; runs++)
{
startTime = DateTime.Now.TimeOfDay;
for (int i = 0; i < iterations; i++)
{
side = random.Next(1, 10);
totalArea = totalArea + (side * side);
}
stopTime = DateTime.Now.TimeOfDay;
TotalTimeOverRuns = TotalTimeOverRuns.Add(stopTime - startTime);
}
Console.WriteLine("Average operation time: " + (TotalTimeOverRuns.TotalMilliseconds /
100).ToString());

//object implementation: run test 100 times and compute average time
Console.WriteLine("");
Console.WriteLine("Execute object implementation:");

TotalTimeOverRuns = new TimeSpan();
for (int runs = 0; runs < 100; runs++)
{
startTime = DateTime.Now.TimeOfDay;
totalArea = new RandomFigureOfSquares(new Number(iterations)
).AreaOfFigure().ToInteger();
stopTime = DateTime.Now.TimeOfDay;
TotalTimeOverRuns = TotalTimeOverRuns.Add(stopTime - startTime);
}
Console.WriteLine("Average operation time: " + (TotalTimeOverRuns.TotalMilliseconds /
100).ToString());
}

class RandomFigureOfSquares
{
public RandomFigureOfSquares(Number numberOfSquares)
{
for (int i = 0; i < numberOfSquares.ToInteger(); i++)
```

```
{
AttachFigure( new Square( new RandomNumberInRange( 1,
10
)
)
);
}
}

private void AttachFigure(Square square)
{
//do something to attach it
this.Area = square.AreaOfSquare();
}

public Number AreaOfFigure()
{
return (Area);
}

Number Area;
}

class Square{
public Square(Number side)
{
this.Side = side;
this.Area = (side * side);
}

public virtual Number AreaOfSquare()
{
return (Area);
}

public virtual Number LengthOfSide()
{
return (Side);
}

private Number Side;
private Number Area;
}

class Number
{
public Number(int number)
{
this.number = number;
```

```
}

public static Number operator +(Number firstNumber, Number secondNumber)
{
return (new Number(firstNumber.number + secondNumber.number));
}

public static Number operator *(Number firstNumber, Number secondNumber)
{
return (new Number(firstNumber.number * secondNumber.number));
}

public override String ToString()
{
return number.ToString();
}

public int ToInteger()
{
return (this.number);
}

protected int number;
}

class RandomNumberInRange : Number
{
public RandomNumberInRange(int min, int max) : base(0)
{
base.number = ( new Random().Next( min,
max
)
);
}

public override String ToString() //probably should have eliminated this
{
return (this.number.ToString());
}
}
```

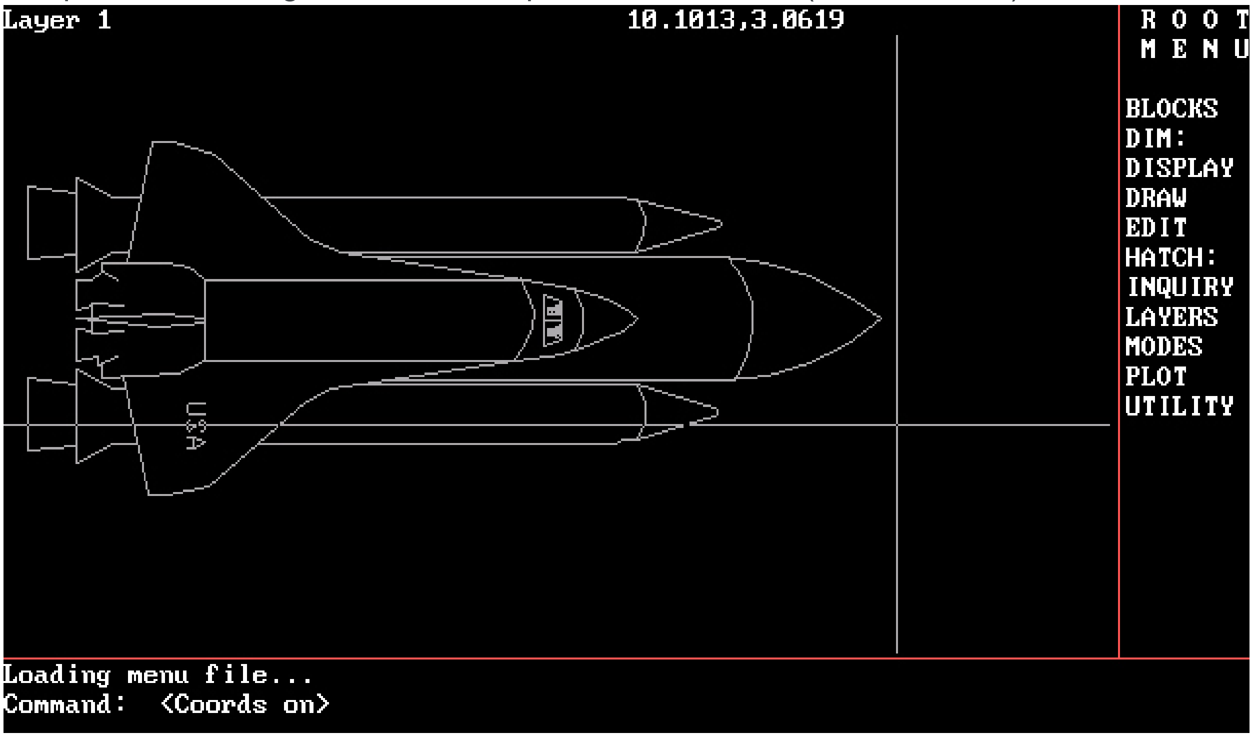
Results are:
Average Time in milliseconds (100 Runs)

Iterations	Log(Iterations)	Objective	Procedural
1,000	3	1.560	0.040
10,000	4	15.531	0.320
100,000	5	154.019	3.080
1,000,000	6	1537.888	29.062

My machine is running pretty slow for some reason so typical results should be better. But, we can see that it takes something like 200,000 iterations for the object oriented approach to become obviously slow, and below 100,00 iterations it's not even a concern. Seriously, who cares if you ask a program to do something and it finishes in 5 hundredths of a second? Human reaction time is about 260 milliseconds so you are actually far too slow to ever notice. More importantly, I don't know why you would even need to do that many iterations of anything 99.99% of the time. I think it most likely would imply a flaw in your design. Furthermore, this test is somewhat biased in favor of the procedural approach since I am testing a simple algorithm rather than a complex one. In reality, using a procedural approach to solve complex problems results in a highly convoluted solution with lots of statements.

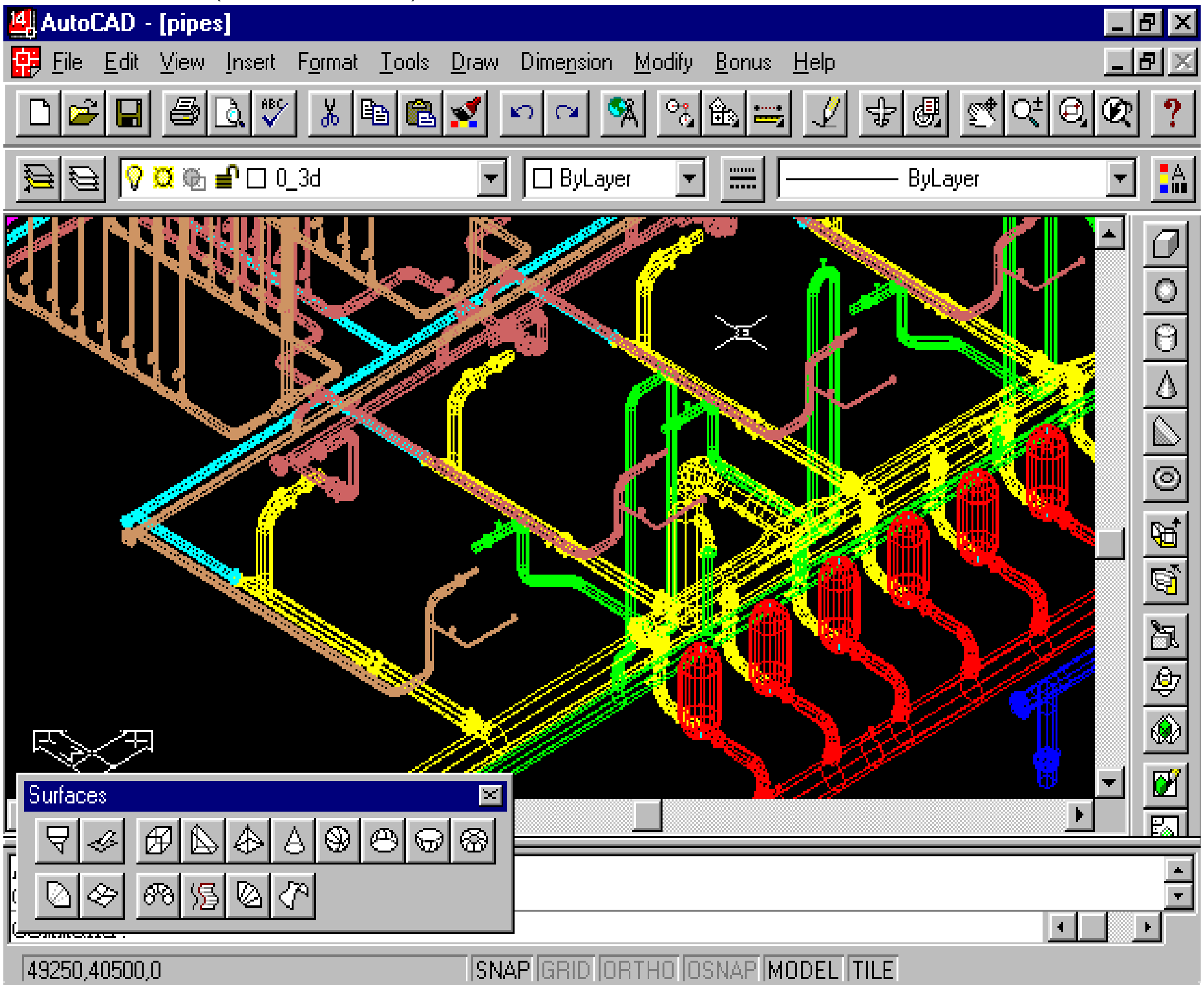
I draw several conclusions from this experiment: (1) For the past 20 years or so, since computers got fast, object simulation has been a viable solution for practically EVERY software system. I think the main exceptions are super high volume programs such as telephone switch systems, and things like scientific and financial systems that require enormous iterations. (2) Object simulation is especially viable when business requirements lead to complex software behavior (e.g. pretty much always). (3) The real impediments to object simulation were primitive computer hardware, and the inclusion of proceduralism in so called OO languages. All of the accusations made by poceduralists are really strawmen.

I want to expand on that point because it's so obvious that you'd have to be brainwashed not to see it. And, this is in fact the case with 95-98% of programmers. Looking from 1935 to 1995 we can see that computer technology was primitive for 60 years. We couldn't make useful computers very useful and very small. They were slow. Graphical systems were poor quality. Equipment was bulky and not very modular. Memory storage was space consuming. The features available in applications were very basic and often buggy (though some would argue that last part hasn't changed much.). The lack of genuine sophistication is demonstrated by complex programs such as AutoCAD - the premier computer aided design software. Compare AutoCAD 2.18 (released 1985) -



```
Loading menu file...
Command:  <Coords on>
```

to AutoCAD 14 (released 1997) -


AutoCAD - [pipes]
File Edit View Insert Format Tools Draw Dimension Modify Bonus Help
0_3d
ByLayer
ByLayer
Surfaces
49250,40500,0
SNAP GRID ORTHO OSNAP MODEL TILE

.

The principle is this: just because a technology is the cutting edge of what we can do doesn't mean it is "advanced". The real meaning of "advanced technology" is that it allows us to do the things we really want to. For example, what good television doesn't have color and isn't a flat screen? Nobody wants a bulky box with a boring black-and-white image, when it really should be a crystal clear window into a color world. In fact, it should be in 3D. Television technology has been very primitive until recently. So, it was absurd to expect an advanced technology like object simulation to outperform more primitive ones on hardware systems that were still learning to "crawl". It's even more absurd to think that it would always be that way given that those hardware systems were improving at an astronomical rate. It was obvious that object simulation would become the new paradigm as soon as computer hardware got sophisticated enough to handle it.

A final thought on a bit of a tangent: do you suppose that object thinking has ever been popular among developers? I don't. In fact, I think they just advanced procedural programming, first by adding structs, then by adding active entities erroneously referred to as "objects". This "objective-procedural programming" is nothing but using these "active entities" pretending to be objects to help you do better procedural programming. That's the closest that 95% - 98% of programmers have ever gotten to object simulation! See what I am getting at? There NEVER WAS A OOP REVOLUTION. The so called paradigm shift to OOP is really a total misrepresentation of object technology by people who have no idea what it is. Since I can't resist, I'd like to introduce another term for this fake object oriented programming: "POOP" - Procedurally Oriented Obsolete Programming. I humbly suggest you might adopt this highly informative and accurate term to clarify the discussion on programming paradigms.

Best,

Zack

• Reply • Share ›

Zack Zack • 2 months ago

EDIT: I wasn't going to paste the actual code here originally because of brevity. However, I changed my mind and posted it because you need to see the code in order to believe the experiment. Now that it's public please don't bother Yegor by asking for my email. You can just paste the c# files into Visual Studio or an interpreter after all.

• Reply • Share ›

Cliff • 2 months ago

Static methods have their uses, such as static factory methods (Bloch, Joshua. Effective Java, 2nd Ed, Item 1). Static factory methods give you more control over object instantiation and emulate the more expressive Ruby "new" you mention in your post.

Overall, though, I agree that these are good guidelines for object design, and they even lend themselves to being more functional in nature (especially immutability). You should **never** do otherwise, but we must mean the pragmatic programmer's "never" in this case: never, except when you need to. Or to quote from the Zen of Python (turn to your hymnals, page PEP20):

> ```
> Special cases aren't special enough to break the rules.
> Although practicality beats purity.
> ```

• Reply • Share ›

Yegor Bugayenko author Cliff • 2 months ago

Yes, totally agree. However, I managed to create an entire web framework without a single public-static method: www.takes.org. This sort of proves that purity can win sometimes that battle :)

1 • Reply • Share ›

Cliff Yegor Bugayenko • 2 months ago

Sure it can. Paul Graham and Robert Morris wrote Viaweb (which became Yahoo! Stores) in pure functional Lisp -- though I can't begin to guess how it was designed. There are plenty of examples of paradigm-purity producing strong results, just as there are **loads** of examples of paradigm-abuse going wrong. Most of the former tend toward functional languages, though, as OOP is abused and bastardized in most non-trivial projects.

That's an impressive feat, by the way. I don't have time to look at it in-depth (and have no plans to use a Java web framework soon) but I skimmed your examples on Github. If I did, though, I'd probably look for something exactly like that.

I do note that Java makes the syntax for OOP-purity pretty verbose in a lot of cases, which is unfortunate. I haven't delved into other JVM languages enough to know if one supports pure OOP in a more terse (but still readable) syntax.

1 • Reply • Share ›

尤川豪 ➦ Yegor Bugayenko • 2 months ago

wow you write a framework to prove your claims. that's impressive! will definitely take a look at it!

• Reply • Share ›

Tony Weston • 3 months ago

Static methods have a use, they mark code that is procedural in nature.... Static (global) variables are different, and shouldn't be used....EVER....

BUT, static methods are fine.....and a stepping stone on the route to creating a good, well designed app. So long as a static's methods dependencies are passed into it as part of its method signature, then everything should work fine, and, it *IS* thread safe by default. Yes, it isn't OO, its procedural... just like C or Cobol. But, you know what, I like that I can SEE it.... its there, in my code... A little bit of hangover procedural, there because I simply haven't thought about my object graph enough to get rid of them.....(Yet!)

When I create a method, one of the first things I think about is if it could be made static..... and if It fits, then I will make it static..... and carry on churning out code.

HOWEVER...... Eventually, these static methods slowly disappear, their interfaces become simpler.... I refactor them away, once I start to understand what my object graph should look like. I can't do this at the start, I just don't know, and I hate doing big design up fronts when I know I will have to change it half way through the project.

WHAT is totally wrong, however, is when I see instance methods masquerading as static

see more

• Reply • Share ›

Matthew Phillips • 3 months ago

Yegor, I've been playing about with writing a tile based map engine and am quite intrigued with this post.
Thinking about my basic classes (or interfaces really).
I would have a tileset which ctor takes an image,
numTilesWide, numTilesHigh, tileWidth and tileHeight. all final and immutable. nice and easy. (This class basically is used for giving me the tile source rectangles for drawing on screen, give it a tileIdx get a rectangle).

Also would have a tile, even simpler ctor has a tilesetIdx and tileIdx, also final and immutable. This allows easy use of multiple tilesets by the engine.

Now the slightly more complex class of MapLayer (suppose our map can have multiple layers, ground level, maybe some dungeons below etc). Its identity would be its heightLevel (0=ground level, 1=first floor, -1=basement/dungeon etc) is this correct (or I suppose could this be correct). It would contain a number of locations (referred to by their x & y coordinates). Its behaviour would be void setLocation(Point coords, Tile tile) and Tile getLocation(Point coords). Is this correct (I know you have getters and setters, how would you handle this?)

Or even more complex, this mapLayer needs saving to and loading from disk (json, xml, csv whatever). As I was previously taught you would have a 'MapLayerLoader' and 'MapLayerSaver' to do this functionality. This clearly goes against your points, how might you handle this? Saving and Loading the state of an object? Should each object know how to save and load itself (breaking single responsibility principle) or some other way?

Not looking for impl code but interested to hear your thoughts

• Reply • Share ›

Yegor Bugayenko author Matthew Phillips • 3 months ago

I'm planning to write a post about that Loader/Saver problem soon. In a nutshell, instead of MapLayerLoader create something like MapLayerInFile and let it behave (!) like a map layer. Don't tell the class to load, but instead let the class behave as if it knows where to get the data. I'll publish a post about it on Monday. And almost the same story with getters and setters. Don't get the data from the object - instead let the object behave the way you want. Make sense?

1 • Reply • Share ›

Matthew Phillips Yegor Bugayenko • 3 months ago

Interesting, i look forward to the article on the loader/saver problem thanks

• Reply • Share ›

Yegor Bugayenko author Matthew Phillips • 3 months ago

Here it is: http://www.yegor256.com/2015/0...

• Reply • Share ›

Goddard Leroy • 4 months ago

Immutability is not the most important thing, the important thing is that objects hold invariants in your code. And a better way to avoid null references is the introduction of sum types.

Interestingly enough, virtue 7 is partially reducing classes in an object oriented language to an organized record of data and functions. I have always wondered, where the hatred for inheritance came from. I guess it is a tooling problem, there are no tools to hold the invariants in your code in most 'modern' object oriented languages.

• Reply • Share ›

Yegor Bugayenko author Goddard Leroy • 4 months ago

Totally agree about invariants. I even mentioned them in this prototype article. We definitely lack them in OOP.

• Reply • Share ›

TryIO • 4 months ago

Hi, Yegor! pretty nice article (I just started to follow you, so I have to recover all your previous posts :P).

I really liked the 7th virtue concept and about that what do you think about the default methods provided by Java8? How can they fit with all of that? If I have in interface with default methods, I still have to extend it in order to have an instantiable class, hence should I leave the default methods in peace? Or maybe there's something wrong with default methods (I have bad feelings about this, but I suspect that...)

• Reply • Share ›

Yegor Bugayenko author → TryIO • 4 months ago

I think this default methods is a replacement of a decorator pattern, but less elegant and flexible. Look at http://github.jcabi.com and its interfaces: http://github.jcabi.com/apidoc... Each interface has a corresponding class `.Smart`, which implements all necessary smart features on top of a default functionality. Every time you need that features, just decorate your class and use them. For example:

```
User user = new Issue.Smart(repo.issues().get(1)).author();
```

`Issue` interfaces returns only JSON, while `Issue.Smart` has method `author()`, which extracts an issue author name from it and instantiates the `User` object. The same could be done with a default method `author()` right in the `Issue` interface, but this would make it bigger and will limit us in the amount of decorators. We'll simply have just one. In a traditional approach, you may have many multi-functional decorators.

So, I'm against default methods and I'm for decorators :)

• Reply • Share ›

Alexander Paderin • 4 months ago

Hi, Yegor.

Please fix typos in one of you snippets:

final class OnlyValidStatus implements Status {
private final Status origin;
public OnlyValidStatus(Status status) {
this.origin = status;
}
@Override
public int read() throws IOException {
int code = this.origin.read();
if (code > 400) {
throw new RuntimException("unsuccessful HTTP code");

```
}
return code;
}
}
```

• Reply • Share ›

Yegor Bugayenko author ➦ Alexander Paderin • 4 months ago

There is no typo, just a constructor omitted. I'll put it there now. Thanks.

• Reply • Share ›

Alexander Paderin ➦ Yegor Bugayenko • 4 months ago

oh sorry, looks like my brain is overheated. I was pretty sure that field type is "String" but not "Status". sorry :)

• Reply • Share ›

Candide Ijon • 5 months ago

Nice post Yegor. I think most of the controversion might come from the fact that you are a bit too strict about statements such as : " you should never use static methods, because their existence violate a few fundamental principles of OOP"
An old article (also resides in a chapter of Effective Java) : http://www.informit.com/articl...
This is a pretty legit use of static methods. Actually there is no "never" in software development (as Pravin Chaudhary stated nicely in the comments). Indeed we have guidelines and how strict we should evaluate them changes from context to context. Strictness for following guidelines may differ a lot for an application developer and a framework/library developer.
In essence nice points and I bookmarked your post :)

• Reply • Share ›

Yegor Bugayenko author ➦ Candide Ijon • 5 months ago

Thanks for bookmarking :) I totally agree that in reality the situations are much more complex and we have to make compromises. However, I believe that my mission, as an author of some "recommendations", is to be as extreme as possible. You, as a reader, will get the idea and will apply it to your reality the best way you can. You, as a practical user of the idea, are allowed to cut corners. I'm not. See my point? :)

1 • Reply • Share ›

Fernando Franzini • 5 months ago

I agree with everything except one thing:

Anthropomorphism is the act of giving life to something that did not exist, going to deal with it a human being has properties and behaviors. Therefore, any designer OO can give life to a class as a Validator, repository or anything else since it becomes a real object. So, to say that "The controller, the parser, the filter, the validator, the service locator, the singleton, or the factory are not good objects..." is not entirely true.

• Reply • Share ›

Yegor Bugayenko author ↪ Fernando Franzini • 5 months ago

"Exists in real life" means that it exists **outside** of the scope of visibility. Parser is something that exists only inside the scope. There is nothing like parser outside the place where we're parsing something. That is what I meant above.

• Reply • Share ›

kmelkez • 6 months ago

Hi,

Nice article, inspiring. But in some case, some principles are difficult to apply. For example, if you work with frameworks like Rails or others, by convention you have some object as ArticleController. Is it a bad naming object for you following the rule "In general, avoid names that end with "-er" — most of them are bad." ?

• Reply • Share ›

Yegor Bugayenko author ↪ kmelkez • 6 months ago

Yes, frameworks like Rails are full of objects that are not "good", according to the article above. This is just sad :(Moreover, the entire design of Rails (and many other frameworks) are very wrong in the first place. About that, in one of the future articles :)

2 • Reply • Share ›

kmelkez ↪ Yegor Bugayenko • 6 months ago

Cool I look forward to reading it !

• Reply • Share ›

valenterry • 6 months ago

I disagree with some of your points.

First is Immutability. That is not a new topic. However, if I am programming a video-game for example, I rather want to have (some) mutable objects. The player for example should be a mutable object. It is counterintuitive to design the player as an immutable object as that is rather different to what we experience in the real world. And I don't even take performance or anything into consideration.

Second is your real world object approach. This is just bad. (I'm sorry for that harsh comment as I really appreciate your blob.)
Even in natural language there are many concepts that are not corresponding to any real world objects. And even though natural language is often a good indicator if something is well designed it is neither always sufficient for IT or mathematics in general nor is it very precise.
I am with you, that one should try to model real-world-entities as you mentioned but there are also real-world-concepts. You should think about this. A filter, btw, IS a real world entity. Like a sand filter. Why would you not extend that real-world-filter to filter other things?

And why must objects be unique? You don't even explain it, you just "believe" it. That is totally fine to tell your reader, BUT your reader then by high chance expects, that you at least give an example which supports your feelings. I don't feel like different but equal http-status-codes are a problem. Rather, in reality we may have two pieces of cloth which can only differentiated by "space&time" (unless you call Navy CIS for forensical analysis...). This "space&time" however can still be checked, as the objects don't have the same identity. But who cares as long as you can interchange them freely (because they are hopefully immutable ;).

I am looking forward to your answer!

• Reply • Share ›

Yegor Bugayenko author ➦ valenterry • 6 months ago

I'm going to write a new article about immutability, where I'll address exactly the concern you expressed. In a nutshell, I believe that the key issue is that in Java, Ruby, JavaScript and many other languages, mutable memory is not accessible as a resource (like we access files, for example). That's why we abuse the object paradigm. I'll write a post about it soon, stay in touch :)

If in your case, a filter is a real-life object, I see no problem with naming a class `Filter`. But this happens very rarely. In most cases, we just give names to objects because this is what they do for us. "*The object will filter my stream of bytes? Aha, it's a filter!*" This is the attitude I'm against.

Good question about uniqueness, I was expecting it. There is no logical answer yet, it's just my feeling for now. I keep thinking about it and sooner or later will write more on this subject. For now, let's say that if an object doesn't have a constructor with arguments, it doesn't have any knowledge coming in. How can an object be *smart* if it doesn't have any input knowledge? See, it's more a philosophical than practical issue.

• Reply • Share ›

valenterry ➦ Yegor Bugayenko • 6 months ago

I don't really get your point about immutability. But that's fine, maybe it will be more clear to me when you write some more detailed post about it. =)
Also, ACK to immutable objects without constructor-arguments. That is of course pretty useless and might be a good case for using enums and similiar.

But, why should I not call an object that filters my stream for something a filter? Okay, properly speaking, it should be a... filterApplier. Uh, that sounds even worse, right. Maybe you have a better suggestion in this concrete-conceptional case?

• Reply • Share ›

Yegor Bugayenko author ➦ valenterry • 6 months ago

Let's say you have an list of files and you want to create a new list, which will contain only text files. Here is how I'm suggesting to do it:

```
List<file> filtered = new FilteredByExtension(list, ".txt");
```

So, it's not a filter, it's a new object, which **is a** list of files, but its properties are different.

• Reply • Share ›

Pravin Chaudhary Yegor Bugayenko • 6 months ago

FilteredByExtension?

You mean you have real life objects named like this?

• Reply • Share ›

Yannick Majoros Pravin Chaudhary • 5 months ago

That's where this made-up stuff FAILS ;-)

• Reply • Share ›

valenterry Yegor Bugayenko • 6 months ago

So you create a new Class for every kind of filter AND of the filtered type (unless they are chainable/decoratable)? Else, you have to put in the logic into the new Object. By that i mean, you would do something like

List<file> filtered = new FilteredByExtension(list, myObjectOrFunctionThatIncludesFilterLogic);
Because now I want to filter for file extension, next for file name. Then I want to filter all file extensions but not these ones. (you can imagine all the different things I wanna do). Everytime a new Class?

• Reply • Share ›

Kanstantsin Kamkou valenterry • 6 months ago

```
$itDirectory = new RecursiveDirectoryIterator(__DIR__);
$itRecursive = new RecursiveIteratorIterator($itDirectory);
$itRecursive->setMaxDepth(2);
$itRegexp = new RegexIterator($itRecursive, '~\.txt$~', Recursive
foreach ($itRegexp as $fileinfo)...
```

• Reply • Share ›

Yegor Bugayenko author valenterry • 6 months ago

Yes, every time you need a filtered list, you create a new object (not a class, an object).

• Reply • Share ›

valenterry Yegor Bugayenko • 6 months ago

No, I mean for every kind of filter, you need a new filter class. E.g., I need a filter, that filters for the first two characters that follow the (last) dot. You need to create a new class for that because the FilteredByExtension class only allows filtering for the complete extension. Unless you forsee that issue and implimented wildcards.

That migh lead to a high number of classes that are only used once or maybe twice at all. And even if that would not be a problem - why would it be better than Filter classes?

• Reply • Share ›

Yegor Bugayenko author valenterry • 6 months ago

In that case you can create a "filtering policy" or a "rule" of filtering, just like you proposed a few messages ago. Just pass than policy into the list that knows how to filter, and you're done. Why it's better than a `Filter`? The article above explains :) An object should be named by its nature, not by the actions he can do for you.

• Reply • Share ›

Yannick Majoros Yegor Bugayenko • 5 months ago

To me, that's just what I call the filter. And it has *no* behavior. Isn't it bad by your own made-up rules?

• Reply • Share ›

valenterry Yegor Bugayenko • 6 months ago

And how would you then name the object that applies the given filteringPolicty?
BTW, how can I format code like you did with "Filter"?

• Reply • Share ›

Marcos Douglas Santos valenterry • 6 months ago

Name `Filter` and `FilterPolicy` :)

1 • Reply • Share ›

Yannick Majoros Marcos Douglas Santos • 5 months ago

And FilterPolicy exists in real life... right ;-)

• Reply • Share ›

Marcos Douglas Santos Yannick Majoros • 5 months ago

Do not?
When we say "exist in real life" does not mean (always) it's a living being but that can be a thing that exists in real life, ie, a filter policy is a thing that exists in real life, right?

• Reply • Share ›

Pravin Chaudhary • 6 months ago

Do you have experience with system software? I feel a lot of your beliefs come from web development.

• Reply • Share ›

Yegor Bugayenko author Pravin Chaudhary • 6 months ago

Most of the software I develop are web apps, you're right.

• Reply • Share ›

• Reply • Share ›

Pravin Chaudhary ➦ Yegor Bugayenko • 6 months ago

Given this, I feel you should qualify your posts to state this. The topics you discuss are mostly generic, so passing sweeping verdicts while only having experience in a subset of the fields (even if the biggest one) is a bit unfair. It could even potentially misguide people. I am surwe you wouldn't want that :)

• Reply • Share ›

Yegor Bugayenko author ➦ Pravin Chaudhary • 6 months ago

I don't see how basic principles of object-oriented programming could be different in web software and in, say, mobile software. But I'm open for corrections and criticism :)

• Reply • Share ›

Pravin Chaudhary ➦ Yegor Bugayenko • 6 months ago

Many of the things you have advocated are not the 'basic' principles of OOP.
e.g., OOP doesn't mandate that all objects be immutable (that's not even possible in many scenarios). Or that 'job titles should no be class names'.

Most of the of good programming practices are not rules; just guidelines. Many of them conflict with each other. Software construction is an art of making tradeoffs to strike a balance. The tradeoffs vary from one area to the other. This is when a web app, a standalone 3D game, a device driver, a compiler - all differ from each other.

5 • Reply • Share ›

Yegor Bugayenko author ➦ Pravin Chaudhary • 6 months ago

I didn't mean that the recommendations listed above *are* the basic principles of OOP. I meant that they are *based* on such principles. And that principles are the same, either in a standalone 3D game or in a web app, I think. Encapsulation is encapsulation, no matter what software you're creating. And static methods are against encapsulation. No matter what software you're writing, either a device driver or a mobile app, you should **never use static methods**, because their existence violate a few fundamental principles of OOP. Of course, we can ignore some of them and use static methods, and our software will work. But this article is trying to promote "a better" way to write OOP programs :)

• Reply • Share ›

Pravin Chaudhary ➦ Yegor Bugayenko • 6 months ago

how about getInstance() method in a singleton class?

• Reply • Share ›

Invisible Arrow ➦ Pravin Chaudhary • 6 months ago

Singleton is a well known anti-pattern due to several reasons. Some of them are:
1. They hide dependencies of your application in your code, instead of exposing them through interfaces. Essentially just global state hanging there for the entire application's life cycle.
2. Objects depending on the singleton instance become untestable as mocking/stubbing won't be possible with static methods.

1 ^ | ˅ • Reply • Share ›

Alex Skorulis • 6 months ago

1. This only applies to model objects. A view model for instance doesn't exist outside your program but is still a valid object.
2. This is nice in principle but can lead to the code being harder to read since you've just doubled the amount of types.
3. Could be better described as always has some non static state. There's nothing wrong with having 2 identical objects.
4. If every object is immutable then how can the state of the app change?
6. The name of a class should tell you what it is. FileReader is an object which reads from a file. Changing the name of this to DataFile does not improve your design, it just makes the purpose of the class more opaque.

4 ^ | ˅ • Reply • Share ›

Fadzlan ➦ Alex Skorulis • 6 months ago

1. Agreed. There are some instances where we can break a way from "real" objects. Even the design community has this kind of debate (skeuomorphic vs flat design)
3.There is nothing wrong having 2 identical objects, but there is nothing wrong not having it too. In this case, I tend to gravitate to minimize the number of objects.
4. This can be done and has been done (ie. Google style guides, Erlang) and is certainly possible. In this case, approaching it from functional perspective helps, which in the end leads to the state being outside of the application (eg. files, database) and data is being passed from function to functions. Of course, this also means that the application does not have any states.
6. If you write a library that is general purpose, I would agree FileReader is a better option. If what you have is specific to your application, I would say that DataFile is more descriptive.

^ | ˅ • Reply • Share ›

Henrique N. do Nascimento • 6 months ago

Yegor, Martin Fowler said that in "26 June 2005".

^ | ˅ • Reply • Share ›

Yannick Majoros ➦ Henrique N. do Nascimento • 5 months ago

And both him and Yegor are just random devs making noise but having no serious backing for their claims.

• Reply • Share ›

尤川豪 Yannick Majoros • 2 months ago

You think Martin Fowler is 'just random devs making noise' ? Seriously?

• Reply • Share ›

Yannick Majoros 尤川豪 • 2 months ago

I think he's no academic source. His rants about "anemic models", for example, are just that. It's just someone with some experience. And some tendency to present things as facts while they are just opinions. If the author wrote books out of his blog posts, that would end up being the same kind of content.

1 • Reply • Share ›

尤川豪 Yannick Majoros • 2 months ago

I see. Thanks for sharing your opinion.

• Reply • Share ›

Yegor Bugayenko author Henrique N. do Nascimento • 6 months ago

And? You think object paradigm was different ten years ago? :)

2 • Reply • Share ›

Gilberto Caetano de Andrade • 6 months ago

Thanks for the article!
One question: what you mean when saying "... once we override the method read() in the child class, ALL METHODS FROM THE PARENT CLASS START TO USE HIS NEW VERSION."? I know we can't insert code in the parent class, that's the idea of inheritance, no?

• Reply • Share ›

Yegor Bugayenko author Gilberto Caetano de Andrade • 6 months ago

Let's say, this is the parent class:

```
class HTTPStatus implements Status, Page {
  @Override
  public int read() throws IOException {
    // original code
  }
  public String html() {
    if (this.read() == 200) {
      // do something
    }
  }
}
```

If you inherit this class and overload method `read()`, you change the behavior of `html()` as well, because in the child class it will call the overloaded version of `read()`. See the problem?

1 • Reply • Share ›

Luca Yegor Bugayenko • 4 days ago

Well, inheritance and method overriding are not bad in themselves.
Troubles occur when you violate Liskov's Substitution Principle, as you do in your example.
It's a matter of discipline.
Your "final or abstract" diktat prevents such problems, though.

• Reply • Share ›

Gilberto Caetano de Andrade Yegor Bugayenko • 6 months ago

More from yegor256.com

yegor256.com recently published

My Favorite Software Books - Yegor Bugayenko

13 Comments Recommend

yegor256.com recently published

Software Quality Award - Yegor Bugayenko

10 Comments Recommend

yegor256.com recently published

How to Protect a Business Idea While Outsourcing

8 Comments Recommend