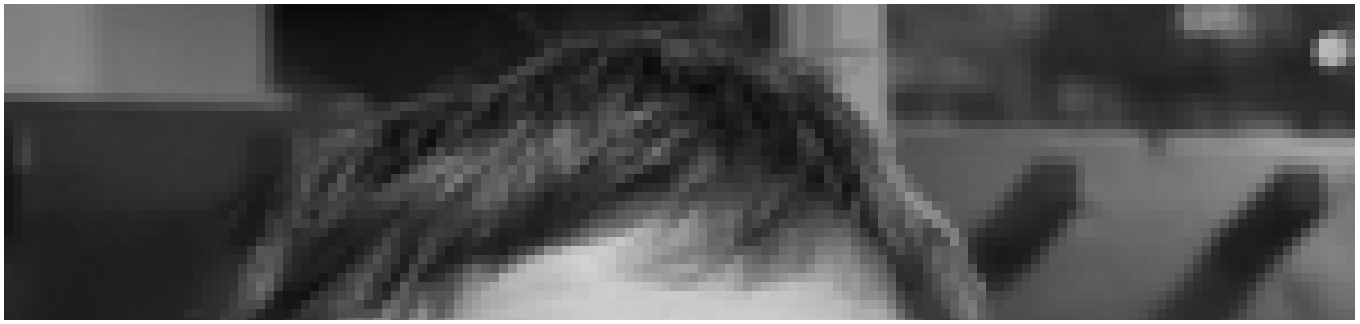
DISQUS Home Inbox Discover Discuss

Community

yegor256.com

83 Comments • Created 7 months ago



An Empty Line is a Code Smell

The subject may sound like a joke, but it is not. An empty line, used as a separator of instructions in an object method, is a code smell. Why? In short, because a method should not contain "parts".

(yegor256.com)

83 Comments

Recommend 1 Share Sort by Newest

Join the discussion…

coder.slynk • 3 months ago

So just remove the empty line. Code smell gone, right? No need to refactor it further.

You don't have a problem with empty lines, you have a problem with methods that do more than one thing. Stop imposing arbitrary rules for yourself. I like to put my local variable declarations at the top of a function and then put an empty line under them, does that mean suddenly I need to put all my local variables into the class scope and make sure they are all thread safe? No, that's stupid. Empty lines in a method are determinant of absolutely nothing.

1 • Reply • Share ›

Leon • 3 months ago

So you're really claiming all empty lines are bad? That's absurd!

I agree with the general idea: if you have blocks of code inside a method that appear to have a common purpose, split them off in a new method.
This isolates scope, encourages code reuse, and auto-documents the code.

That said, you need to be pragmatic about it. Just because you can split up your twenty line method in ten methods with two lines, doesn't mean you should. That smells either of over engineering or OCD.

Finally, I quite like my empty lines, I would have added several in your "new class" too (especially between the class members). They help me visually identify code blocks (as in: separate methods or loops) and features, and quickly see when something is "off". I know this is a personal preference, but my code style is based on years of tweaking to improve readability.

Btw, it appears you should have posted a new article per paragraph, because there are empty lines between them ;-) .

1 ▲ ▼ • Reply • Share ›

Oleg Majewski • 4 months ago

what about

```
import static com.google.common.base.Preconditions.checkNotNull;
import static javax.annotation.Nullable;

public class MyClass {
  ...

    public void doSomething(String first, String second, @Nullable String last) {
        checkNotNull(first, "first");
        checkNotNull(second, "second");

        // start doing stuff after params check
    }
}
```

see also https://github.com/google/guic...

▲ ▼ • Reply • Share ›

Yegor Bugayenko author ➚ Oleg Majewski • 4 months ago

Like in the example below in the comment thread, you should not check params and do other stuff in the same method. These are two different activities. Create a decorator of the class and call it SafeMyClass, which will check input parameters and call similar methods in the encapsulated object.

▲ ▼ • Reply • Share ›

Oleg Majewski ➚ Yegor Bugayenko • 4 months ago

looking at one of your own classes: https://github.com/jcabi/jcabi...

well, in real world you seems not to follow your own rules, you have correctly java docs and empty lines between methods for example. You also seperate imports and package etc.

Looking at one of the methods:

```
/**
 * This class is immutable?
 * @param type The class to check
 * @throws ImmutabilityChecker.Violation If it is mutable
 */
private void check(final Class type)
    throws ImmutabilityChecker.Violation {
    synchronized (this.immutable) {
        if (!this.ignore(type)) {
            if (type.isInterface()
                && !type.isAnnotationPresent(Immutable.class)) {
                throw new ImmutabilityChecker.Violation(
                    String.format(
                        "Interface '%s' is not annotated with @Immutable"
                        type.getName()
                    )
                );
            }
            if (!type.isInterface()
                && !Modifier.isFinal(type.getModifiers())) {
                throw new Violation(
                    String.format(
                        "Class '%s' is not final",
                        type.getName()
                    )
                );
            }
            try {
                this.fields(type);
            } catch (final ImmutabilityChecker.Violation ex) {
                throw new ImmutabilityChecker.Violation(
                    String.format("Class '%s' is mutable", type.getName()
                    ex
                );
            }
            this.immutable.add(type);
            Logger.debug(this, "#check(%s): immutability checked", type);
        }
    }
}
```

Well, hehe, you don't have empty lines, but that check method could/should be spitted in smaller chunks, e.g. like this:

```
 /**
 * This class is immutable?
 * @param type The class to check
 * @throws ImmutabilityChecker.Violation If it is mutable
 */
private void check(final Class type)
    throws ImmutabilityChecker.Violation {
    synchronized (this.immutable) {
        if (!this.ignore(type)) {
            ensureClassIsAnnotatedWithImmutable(type);
            ensureClassIsFinal(type);
            try {
                this.fields(type);
            } catch (final ImmutabilityChecker.Violation ex) {
                throw new ImmutabilityChecker.Violation(
                    String.format("Class '%s' is mutable", type.getNa
                    ex
                );
            }
            this.immutable.add(type);
            Logger.debug(this, "#check(%s): immutability checked", ty
        }
    }
}
private void ensureClassIsFinal(Class type) throws Violation {
    if (!type.isInterface()
        && !Modifier.isFinal(type.getModifiers())) {
        throw new Violation(
            String.format(
                "Class '%s' is not final",
                type.getName()
            )
        );
    }
}
private void ensureClassIsAnnotatedWithImmutable(Class type) throws V
    if (type.isInterface()
        && !type.isAnnotationPresent(Immutable.class)) {
        throw new Violation(
            String.format(
                "Interface '%s' is not annotated with @Immutable",
                type.getName()
            )
        );
    }
```

```
    }
```

What I want to say is: totally agree to "separation of concerns" goal, but there is nothing bad about empty lines in general ;-)

2 ▲ | ▼ • Reply • Share ›

Oleg Majewski ➔ Yegor Bugayenko • 4 months ago

why, this class will never be used without that check. Actually the check is part of the contract of that class: if you give me null values, it's your fault, not mine.
Also the interface is exactly the same.
Creating a whole class, delegating to origin, override a method all only for 2 lines of checking sounds like a lot of boilerplate.
To count this in lines: you need to create at least 1 line package, 2 lines class, 2 lines method, 1 line origin field, 3 lines ctor ... to omit 1 single empty line.
Can you provide a real example how what you suggest is helpful?

▲ | ▼ • Reply • Share ›

Yegor Bugayenko author ➔ Oleg Majewski • 4 months ago

"this class will never be used without that check" - how do you know? You create a class and your clients decide how to use it. If the client need it to be super safe, he will wrap the class into another decorator. If the client is confident that the input is always correct, he doesn't use the input-checking decorator.

Keep in mind, this article is about empty lines **inside** methods, which are code smell.

▲ | ▼ • Reply • Share ›

Oleg Majewski ➔ Yegor Bugayenko • 4 months ago

still missing a real world example, do you have any?

> "this class will never be used without that check" - how do you know?

the method declared it:

```
public void doSomething(String first, String second, @Nullable Str
    checkNotNull(first, "first"); // don't give me null
    checkNotNull(second, "second"); // and here not as well
}
```

> You create a class and your clients decide how to use it. If the

> You create a class and your clients decide how to use it. If the client need it to be super safe, he will wrap the class into another decorator. If the client is confident that the input is always correct, he doesn't use the input-checking decorator.

hehe, sure, maybe in a dream. But you know that client which might be "confident" potentially have bugs, and some client WILL 100% SURE put a wrong value in such a method with a not null param at some time. What now? Lets your code for an example:

```
final class TextFile {
    private final File file;
    TextFile(File src) {
        this.file = src;
    }
    public int grep(Pattern regex) throws IOException {
        return this.count(this.lines(), regex);
    }
    private int count(Iterable<string> lines, Pattern regex) {
        int total = 0;
        for (String line : lines) {
            if (regex.matcher(line).matches()) {
                ++total;
            }
        }
        return total;
    }
    private Iterable<string> lines() throws IOException {
        Collection<string> lines = new LinkedList<>();
        try (BufferedReader reader = new BufferedReader(new FileRe
            while (true) {
                String line = reader.readLine();
                if (line == null) {
                    break;
                }
                lines.add(line);
            }
            return lines;
        }
    }

    public static void main(String[] args) throws IOException {

// my buggy client, it can be of course much more complex
        TextFile file = new TextFile(null); // here is the bug, my
        file.grep(null); // it may not put nulls here
    }
}
```

```
}
```

The stacktrace executing the main now looks like:

```
Exception in thread "main" java.lang.NullPointerException
        at java.io.FileInputStream.<init>(FileInputStream.java:124
        at java.io.FileReader.<init>(FileReader.java:72)
        at org.example.TextFile.lines(TextFile.java:30)
        at org.example.TextFile.grep(TextFile.java:17)
        at org.example.TextFile.main(TextFile.java:44)
```

Where is the bug? Who is responsible? main? grep? lines? Actually that Nullpointer is thrown from java.io.FileInputStream. This is a simple example, think about a stacktrace with 50 lines going over > 3 3rd party libs. ;-)

This can be fixed simple:

```
public int grep(Pattern regex) throws IOException {
    Preconditions.checkNotNull(regex);
    // the evil empty line, cause I like to seperate param checks
    return this.count(this.lines(), regex); // here it is just one
}
```

the stacktrace is more clean now:

```
Exception in thread "main" java.lang.NullPointerException
        at com.google.common.base.Preconditions.checkNotNull(Preco
        at org.example.TextFile.grep(TextFile.java:19)
        at org.example.TextFile.main(TextFile.java:48)
```

It is pretty clear that in grep(TextFile.java:19) the precondition check cause the error, thus something is wrong with main(TextFile.java:48).

The last improvement would be to add that precondition check also to the constructor, but you got it already ;-)

To sum up: NO, I don't want to allow any client to decide, if it use my class in a safe or not safe mode, MY class and MY method is ALWAYS safe.

▲ | ▼ • Reply • Share ›

Tomasz Górecki • 4 months ago

I don't think this would be applicable in non-OOP paradigms. Or maybe you do have some idea what it could like?

▲ | ▼ • Reply • Share ›

Yegor Bugayenko author → Tomasz Górecki • 4 months ago

I think, in most languages an empty line inside a method body is a code smell, but in OOP especially. What language/paradigms do you have in mind? Let's discuss a specific example.

• Reply • Share ›

Tomasz Górecki → Yegor Bugayenko • 4 months ago

I'm thinking about C. Consider the sample code (please ignore issues such as naming, not using const etc., not a real-life code):

```
void func(int *a, int *b) {
    if (a == NULL) { return; }
    if (b == NULL) { return; }

    do_things(*a, *b);
}
```

In case of C, using pointers does make sense, and they may be NULLs. Checking for that is a must (or we'd occassionally get segfaults, not nice). In this example I think it's quite reasonable to use empty lines.

• Reply • Share ›

Yegor Bugayenko author → Tomasz Górecki • 4 months ago

Good example indeed. C language is rather difficult to call a programming language. It's more like a scripting instrument. If it would be C++, I would say that you should create another class, a decorator, which should wrap your existing class and check input values for NULL. But in C... I don't really know what to suggest :) So, in this case, an empty line is a code smell. But it's not that particular code that smells, but the entire programming language.

1 • Reply • Share ›

Tomasz Górecki → Yegor Bugayenko • 4 months ago

Aw, this seems to be rather opinion-based :) C's for scripting? I couldn't disagree more. It definitely is a proper language.
This way you dismiss all non-OO languages as smelly. OOP has it's pros, but it's not the only possible approach.

Majority of your posts concerns OOP, but it's not used everywhere. Do you think it should? Might be a good topic for one of your articles: 'Why OO is the best paradigm' or something like that.

Anyway, could you please show us how you'd do this in C++, using a decorator?

• Reply • Share ›

Yegor Bugayenko author → Tomasz Górecki • 4 months ago

Here is how I would do this in C++:

```
#include <ctime>
class Foo {
  public:
  virtual void func(int *a, int *b);
};
class BaseFoo : public Foo {
  public:
  virtual void func(int *a, int *b) {
    // do_things(*a, *b);
  }
};
class SafeFoo : public Foo {
  public:
  SafeFoo(Foo& foo) : origin(foo) {
    this->origin = foo;
  }
  virtual void func(int *a, int *b) {
    if (a == NULL) { return; }
    if (b == NULL) { return; }
    this->origin.func(a, b);
  }
  private:
  Foo& origin;
};
```

And then:

```
Foo foo = new SafeFoo(new BaseFoo());
foo.func(a, b);
```

1 ▲ | ▼ • Reply • Share ›

Marcos Douglas Santos ➦ Yegor Bugayenko • 4 months ago

Yes.

Implementing so you can test BaseFoo without using tests with NULL and can test SafeFoo using a fake BaseFoo too.

▲ | ▼ • Reply • Share ›

Andrej Istomin • 4 months ago

Today I saw code written by guy who probably likes double-spaced intervals in Word document: almost every line of his code followed by empty line :) And, yes, of course, logical parts in his 'transaction scripts' separated with 2 and even 3 empty lines :)

▲ | ▼ • Reply • Share ›

Yegor Bugayenko author ➦ Andrej Istomin • 4 months ago

Did you give him a link to this article? :)

▲ | ▼ • Reply • Share ›

Andrej Istomin ➚ Yegor Bugayenko • 4 months ago

Sure :)

^ | v • Reply • Share ›

Grzegorz Gajos • 7 months ago

I was looking for example of code that is exception of this rule and I realized that well known approach to code testing is flawed! Who is not separating blocks in given/when/then approach :). Especially when amount of lines in blocks growing.

```
@Test
public void Test() {
  // given
  theGivenInput();

  // when
  theServiceIsCalled();

  // then
  resultIsCorrect();
}
```

I realized also that it can be easily encapsulated in OO approach:

```
@Test
public void Test() {
  new BDD()
    .given(new TestInput())
    .when(new TestBehaviour())
    .then(new TestAssertions());
}
```

2 ^ | v • Reply • Share ›

Джек ➚ Grzegorz Gajos • 5 months ago

And now you have 10 times more lines in other places. Don't be crazy about OOP - it is just a tool to make a computer do what you want.

1 ^ | v • Reply • Share ›

Grzegorz Gajos ➚ Джек • 5 months ago

And it's x times more flexible. You can reuse test input, behaviour and output because you're not coupling things together.

1 ^ | v • Reply • Share ›

Джек ➚ Grzegorz Gajos • 5 months ago

In case if you need that flexibility. Otherwise it is just overcomplication.

1 ^ | v • Reply • Share ›

Grzegorz Gajos ➚ Джек • 5 months ago

Hard to argue about few lines of not existing software. I made small JavaScript lib lately https://github.com/ggajos/ot-j... using OO tests. Thanks to this I could use test objects to generate readable documentation also.

 • Reply • Share ›

Джек ➚ Grzegorz Gajos • 5 months ago

Your words sound like premature optimization for me. KISS.

1 • Reply • Share ›

Zhao Zhiming ➚ Grzegorz Gajos • 7 months ago

yes, this is what I mean.

 • Reply • Share ›

Grzegorz Gajos ➚ Zhao Zhiming • 7 months ago

True, I overlook your comment. I totally agree with you.

 • Reply • Share ›

Invisible Arrow ➚ Grzegorz Gajos • 6 months ago

I currently use Mockito for unit testing in Java, which has the style in the first snippet (having blank lines separating mocks/calls/assertions/verifications).
Was curious to know which is the framework that supports the BDD style in Java as mentioned in the second snippet?

 • Reply • Share ›

Andrej Istomin • 7 months ago

Hehe... hot discussions as usual :) Let me add my two cents here. I was surprised as many other people that you don't use empty lines to separate code blocks. I was think a lot about it and that is my opinion. I would compare this approach of using empty lines with using sticks when somebody's legs are ill. If you have normal health you don't need the stick. If your code is good designed - you don't need empty lines. Now for me it's one of the first indicator when my code begins smelling. As I see that I need to separate something with empty lines -> I'm sure that this piece of code needs refactoring. Unfortunately in many projects people don't have time for this refactoring and reality is that sometimes we have to accept it. Yes, our legs are ill sometimes... but it's not an option to say that human being can not walk without a stick :)

 • Reply • Share ›

Yegor Bugayenko author ➚ Andrej Istomin • 7 months ago

It's a very good metaphor! Indeed, if your code is ugly, either admin it or try to fix it. But don't say that it's beautiful. If your legs are ill, use the stick. But don't tell everybody that your legs are perfectly healthy and everybody should use the stick :)

2 • Reply • Share ›

Zhao Zhiming • 7 months ago

I think sometimes must have empty lines in method, like unit test, it general prepare test

data first, then execute method, and verify the execute result at last, so it must have 2 empty lines in a unit test method at least.

• Reply • Share ›

Yegor Bugayenko author → Zhao Zhiming • 7 months ago

Test methods, as well as non-test ones, must be cohesive and must do one thing. If you see that there are three blocks of code in your test method, try to use supplementary methods or classes, which will help you to break your scope down into pieces. Empty lines is not a solution, no matter what method we're working with.

2 • Reply • Share ›

Andrej Istomin → Yegor Bugayenko • 7 months ago

I think it's good reason for you to write new article about unit tests ;) Many developers think that unit tests are not a code, but 'something special'. That's why maybe in many projects unit tests are so ugly and unreadable. Maybe these developers think that unit tests is something 'one-time useful'(in despite of they 100% know that it's not true)... I don't know how to explain this self-forgiveness about tests: my code is ugly? oh, it's just tests... Normally, looking at project's tests I want to understand how code works. But often tests are more unreadable than code itself. It's really a big mistake in my opinion. I think that tests' quality must not be worse than production code quality, maybe must be even better.

1 • Reply • Share ›

Yegor Bugayenko author → Andrej Istomin • 7 months ago

Yes, that's exactly what is happening. Test methods are considered as something less important than the main code. And this is a defect of our programming languages - they are not **designed for unit testing**. Java doesn't know anything about test methods or test classes. But it should. Until then, we should be as careful and attentive with our test code as we are with the main code.

• Reply • Share ›

Kanstantsin Kamkou • 7 months ago

Yegor, could you throw some light on my conclusion: this topic is valid only in case of standalone classes/interfaces/traits/etc. and not valid for controllers (the place where we're trying to connect multiple logic together)?

p.s. the first filing related to the CSS example is that `.wide .important .danger` tells me nothing. In the same time, h1..n works. A little bit better situation is with `.wide .huge .danger`. With `box-shadow` and prefixes it is more transparent I think.

• Reply • Share ›

Yegor Bugayenko author → Kanstantsin Kamkou • 7 months ago

I think it's relevant to any class. Why controller is an exception? BTW, a controller (or manager, or validator, or dispatcher) is an anti-pattern by itself, see http://www.yegor256.com/2014/0... But even if you have to use controllers, they should be formatted with the same rules in mind as any other class. Why not?

should be formatted with the same rules in mind as any other class. Why not?

• Reply • Share ›

neilstockton • 7 months ago

There are actually studies that conclude very clearly that the eye and brain can process information better when separated by spaces. I'll have blank lines in my code thanks very much, and I'll structure my methods around what they are intended for and if that means there are two operations to achieve that aim then so be it.

Somebody comes on and decides what they do is right and what everyone else does is wrong; just another day in the opinionated programming world where egos are the only thing that matters.

5 • Reply • Share ›

Lukas Eder • 7 months ago

I think, this blog post of ours has just reached its peak in terms of relevancy:

http://blog.jooq.org/2014/07/25/top-10-very-very-very-important-topics-to-discuss/

4 • Reply • Share ›

ARayblt ➚ Lukas Eder • 7 months ago

While the specific issue here is just silly, software development's infatuation with simplistic metrics and just-so stories about how to improve programming is of some interest. The non-sequiturs offered here are fascinating

2 • Reply • Share ›

Jason Reid • 7 months ago

Personally, I think that this is a relatively inconsequential refactoring. It neither narrows the class' interface nor eliminates any dependencies. In fact, you've increased the total number of lines of code without eliminating a byte of complexity from the work the class is doing.

The difference between the two Java examples is a few highlighted lines and an automated "extract method" refactoring. I don't see a significant improvement.

1 • Reply • Share ›

Nathan Green • 7 months ago

I think I spent several days last week trying to refactor code to get rid of all the spacer lines. Instead of one 450-line method, there are now a couple dozen 10-20 line methods. I didn't get rid of all of the spaces, but at least now it's *possible* to write some unit tests. It's possible to test methods that do two or three things, even though that's wrong. A method that does 300 things is practically impossible to test, and impossible to read without empty lines.

Following this rule might require a little more up-front work, but the maintenance difficulty will be so much lower. When you think about it, it's obvious: if you have a 5 line method that does the wrong thing, it's cheap to write a brand new one that works correctly. But if you have a 500 line method that does the wrong thing, you're stuck with it.

1 • Reply • Share ›

ARayblt ↗ Nathan Green • 7 months ago

Are you saying the blank lines helped you refactor the code?

^ | v • Reply • Share ›

Nathan Green ↗ ARayblt • 7 months ago

Yes and no. Sometimes they were helpful and sometimes they got in the way. Maybe half of the time they were used to delimit independent functional concerns that could be refactored into distinct methods. Many seemed to be there just to relieve eye strain from page after page of imperative code, with no obvious structural grouping at all.

^ | v • Reply • Share ›

Yegor Bugayenko author ↗ Nathan Green • 7 months ago

I'm glad you see the effect of this rule. You can go even further and try to use Qulice, which will strictly prohibit all empty lines in all methods. We created a custom Checkstyle check for it: http://www.qulice.com/qulice-c...

1 ^ | v • Reply • Share ›

Nathan Green ↗ Nathan Green • 7 months ago

I looked over a parser I wrote by hand and realized it had no blank lines. It was quite readable, despite tracking multiple mutable variables. Of course you'll probably say I shouldn't be using mutable state, but I've never written a parser any other way.

^ | v • Reply • Share ›

Kneck • 7 months ago

Most stupid programming-related post ever.

4 ^ | v • Reply • Share ›

Patrick Sagan • 7 months ago

Great topic. It's very good practice to think about new method when putting new line in current method to separate that "micro-logic" :) It's more like psychological aspect of coding to learn new, smart habit, because often that empty lines are used to separate "micro-logic" in one method breaking single responsibility principle for many reasons like general weak code design, laziness, don't being aware of single responsibility principle etc. In my opinion it's good to think about this rule every time clicking enter in method to separate something.

^ | v • Reply • Share ›

Brian • 7 months ago

This is the most RIDICULOUS thing I have ever read. Utterly stupid. You don't know what you're talking about, and should stop acting as though you do. Grow older; gain actual wisdom; post things like this later.

3 ^ | v • Reply • Share ›

Yegor Bugayenko author ↗ Brian • 7 months ago

Mom?! Are you reading my blog?

13 ^ | v • Reply • Share ›

ARaybIt • 7 months ago

If the reasoning here makes any sense at all, then you should be limiting all methods to one expression each.

2 ^ | v • Reply • Share ›

Yegor Bugayenko author ↗ ARaybIt • 7 months ago

Ideally, yes

^ | v • Reply • Share ›

ARaybIt ↗ Yegor Bugayenko • 7 months ago

Then you would end up with a huge number of methods that are only called once, with no realistic prospect of reuse, because their only purpose is to be an expression in a larger algorithm, and their only justification for existence as methods would be a simplistic and dogmatic interpretation of what Single Responsibility means.

7 ^ | v • Reply • Share ›

neilstockton ↗ ARaybIt • 7 months ago

Just imagine the stack trace you could get with methods that small. Would make JBoss traces seem small. LOL

1 ^ | v • Reply • Share ›

David Bowman ↗ ARaybIt • 7 months ago

Micro-methods! New buzz word, I love it!

^ | v • Reply • Share ›

Bartek Andrzejczak ↗ ARaybIt • 7 months ago

Method is not just some reusable piece of code. It wraps some actions and gives them the name, which should tell you what does it do. So if you use some conditions just once, but it's complicated, you should put it into separate method and call it, so it can tell you what does it do without demanding some time to understand algorithm behind it.

4 ^ | v • Reply • Share ›

ARaybIt ↗ Bartek Andrzejczak • 7 months ago

What you say is true in general, but I think you are overlooking the context of my comment. You write 'if you use some conditions just once, but it's complicated, you should put it into separate method', but I am writing about a situation where each method contains a single expression, and, furthermore, where every single distinct expression in the program is in its own method.

If you think this is ridiculous, it is - the point is that it is the reductio ad absurdum in an argument that Yegor's reason for deprecating blank lines is flawed. If a blank line is an indication that a method has violated the Single Responsibility principle, or 'has parts', as Yegor

puts it, then an even better indication would be that it contains more than one expression (after all, a blank line is not a part at all, and taking a blank line out does not turn two parts into one.) Yegor agreed (perhaps too hastily), hence my explanation of what it would lead to.

Yegor approves of Bob Martin's rule 'a properly designed method [may] have up to ten instructions'. What single thing would you be doing with ten instructions (we're talking OO, not assembler, here, so I will assume 'instructions' are expressions)? One very common pattern is initialization followed by iteration, and neither on its own completes a single task. Putting a blank line between them helps a little in reading the code, and it certainly cannot alter the cohesion of the method, because cohesion is a semantic issue, and a blank line isn't even syntax - it's just typography.

Yegor follows by writing 'Empty lines inside methods encourage us to break this awesome rule and turn them into multi-page poems.' Really? I am pretty sure that it is rare for a coder to think 'I want a blank line here, so I will have to come up with some extra code to follow it.' I think it is much more likely that programmers write large, low-cohesion methods either because it is expedient to do so, or because they have difficulty in thinking about their program in somewhat abstract terms such as cohesion and responsibilities.

This proposal is just one of many that attempts to use a simplistic measure of syntax (or something even weaker, in this case) as a proxy for semantic issues that are not easily measured (Martin's 'rule of ten' is another, though not so bad.) Let us suppose, for the sake of argument and however unlikely, that there was some correlation between blank lines and violations of the Single Responsibility principle. Once you make an issue of it, however, programmers will change their behavior, invalidating the statistical basis of the metric. In particular, no programmer who lacks the abstract thinking skills and judgement to design good code will magically acquire these skills; instead, he will just stop putting in blank lines. This is what happens whenever you use a simplistic metric as a proxy for a difficult problem: you don't fix it, you just hide the evidence of it.

1 ▲ | ▼ • Reply • Share ›

Grzegorz Gajos ➚ ARayblt • 7 months ago

Ok, but code smell isn't evidence of error. It might be an error and in most cases it is. However, can you provide example when empty line in middle of method is used for proper reason?

▲ | ▼ • Reply • Share ›

ARayblt ➚ Grzegorz Gajos • 7 months ago

'...in most cases it is [an error]'. If 'it' is referring specifically to blank lines, I think you just made that up, and if you did not, please show the evidence

the evidence.

The burden of evidence lies on the people making that claim, and that evidence has to be statistical in nature (a highly significant correlation, given that you used the word 'most', and furthermore, one that is not merely explained (in the statistical sense) by a correlation with a primary factor with more explanatory power, such as the number of statements.)

A 'just so' story with an example appended does not cut it - engineering uses statistical methods, stories are for amateurs.

1 ▲ | ▼ • Reply • Share ›

Grzegorz Gajos → ARayblt • 7 months ago

Can you please provide example of method or function where blank space is used for good reason? You can prove that this theory is wrong/false by just providing single example that's wrong. Just like in mathematical world. No need statistics here I'm afraid.

▲ | ▼ • Reply • Share ›

ARayblt → Grzegorz Gajos • 7 months ago

You have contradicted yourself:

'[A] code smell isn't evidence of error. It might be an error and in most cases it is.'

'No need [for] statistics here'

The first claim is inherently statistical, so first things first: do you consider this to be a statistical issue, or not?

1 ▲ | ▼ • Reply • Share ›

Grzegorz Gajos → ARayblt • 7 months ago

Thank you for corrections, indeed I made that up. It was my personal opinion and it's pointless to argue about that. Just thought you can provide example of function where blank space is used for good reason. It's pure curiosity because I have no idea when it's justified.

▲ | ▼ • Reply • Share ›

ARayblt → Grzegorz Gajos • 7 months ago

OK - when I saw you claiming that a single counter-example could settle the matter, I thought you were trying to shift the burden of evidence away from the proponents of Yegor's rule.

That rule actually has two parts: firstly that blank lines are useful indicators of a violation of the Single Responsibility rule, and secondly that banning them will noticeably reduce those violations.

As I wrote before, blank lines are merely typography, so their use (or not) is always optional and somewhat subjective. Note that the Single

Responsibility rule is also somewhat subjective, because responsibilities are hierarchical and sometimes cross-cutting, and where you draw the line is not entirely objective.

Several people have said here that blank lines can make code easier to read. You may believe that any example would be better if split into multiple methods, but there is such a thing as too much fragmentation, as I pointed out in an earlier reply - you end up with many methods that have less than one responsibility. I have traced through too many methods looking for where something actually gets done to regard this as a hypothetical problem.

I gave an example of one common pattern where a blank line may help with readability - initialization followed by iteration. Because initialization sets up the preconditions for the loop invariant, they are both part of a single responsibility.

What are probably the best examples I have seen of the helpfulness of blank lines are no longer available to me, and would not be particularly helpful out of context anyway - they involved the modeling of nuclear physics. Here, however, is a simple example that I think is slightly more readable with blank lines:

```
    private static String neutralize(final String item) {
        final int start = item.indexOf('#');
        final String result;

        if (start == 0 || start > 0 && item.charAt(start - 1) != '
            && !DockerRun.inquotes(item, start)) {

            result = new StringBuilder(item.substring(0, start))
                .append('`')
                .append(item.substring(start))
                .append('`')
                .toString();
        } else {
            result = item;
        }
        return result;
    }
```

• Reply • Share ›

Grzegorz Gajos ↗ ARayblt • 7 months ago

Thanks for snippet. I totally agree with you. Removing blank lines from this code makes it less readable. Moreover it doesn't make sense to extract more methods because variables are accessed across multiple lines. It will make code less readable. However,

something is wrong with this method body:
- it's private static method which means it doesn't belong to any class at all,
- it's not testable and not reusable in other places,
- if statement conditions was already so complex that was extracted to DockerRun.inquotes?
I believe that this method should be refactored to object (draft implementation):

```
class Neutralizer {
  String item;
  int start;
  Neutralizer(String item) { this.item = item; item.indexOf('#');
  boolean isStartingWithHash() { return start == 0 }
```

see more

• Reply • Share ›

Cliff ➚ Grzegorz Gajos • 2 months ago

<action>-"er" is usually considered a code smell these days as well. It's often indicative of the functional decomposition antipattern, anemic models, or god objects. I'd argue the first case here.

Even the author here recommends against it (in an admittedly more recent article, but he's not the first to say so):
http://www.yegor256.com/2015/0...

• Reply • Share ›

ARayblt ➚ Grzegorz Gajos • 7 months ago

Personally, I do not think the issues you raise amount to a significant criticism of this code, but perhaps you could raise them with the code's author - it is taken from the project that Yegor presented to me as 'clean, good looking, cohesive and easy to read' in an earlier post here. You can find a link to the project in that post.

• Reply • Share ›

Grzegorz Gajos ➚ ARayblt • 7 months ago

I agree that it is not very significant change, however, coding is all about small, not very significant changes. I like Uncle Bob advice "extract till you drop". **@Yegor Bugayenko** what do you think?

• Reply • Share ›

ARayblt ➚ Grzegorz Gajos • 7 months ago

So you are saying that a piece of code presented by Yegor as evidence that his rule works does not, as you see it, properly conform to the Single Responsibility rule. I will leave it to the two of you to fight that one out.

• Reply • Share ›

Grzegorz Gajos ↗ ARayblt • 7 months ago

@Yegor Bugayenko @ARayblt; Im my opinion neutralize method is violating single responsibility rule. You can see into how many cases it can be splitted in refactored Neutralizer class. This I believe confirms that empty line is indeed code smell. Even if there is no empty line but it feels right to put one there.

^ | v • Reply • Share ›

ARayblt ↗ Grzegorz Gajos • 6 months ago

'...Even if there is no empty line but it feels right to put one there.'

So now imaginary blank lines are a code smell? **@Yegor Bugayenko**, perhaps you should add them to your qulice rules.

^ | v • Reply • Share ›

Yegor Bugayenko author ↗ Grzegorz Gajos • 7 months ago

I think that when an empty line is making the code more readable - it's a bad sign. A very good metaphor were made by **@Andrej Istomin** above - bad design is like a decease. You can fix it temporarily with a few empty lines and make it look a bit more readable, but it doesn't cure the decease, only make it worse. You should fix the code instead of patching it with empty lines. That how I see it at least :)

^ | v • Reply • Share ›

Yegor Bugayenko author ↗ ARayblt • 7 months ago

Keep in mind that statistically speaking Nokia was the best and the most popular phone five years ago. My point is that if something is more useful it doesn't mean that it's the best. And in software industry, in most cases, it means exactly the opposite.

^ | v • Reply • Share ›

ARayblt ↗ Yegor Bugayenko • 7 months ago

If your comment about Nokia had any relevance here, it would apply equally to almost every other use of statistics - another reductio ad absurdum.

Secondly, I am not arguing that your coding rule is not quite optimal, I am claiming that there is no evidence that it is useful at all, and good reason to suspect that it is not.

1 ^ | v • Reply • Share ›

Yegor Bugayenko author ↗ ARayblt • 7 months ago

Look at this piece of code: https://github.com/yegor256/ru... I find it clean, good looking, cohesive and easy to read. Don't you agree?

^ | v • Reply • Share ›

ARayblt ↗ Yegor Bugayenko • 7 months ago

That doesn't help at all, because you cannot build a general case out of examples.

PS: Another point: for every example of a substandard program your rule would catch, I can show you an equally bad program that it would let through (I am sure you know why I can be certain of this.) In other words, I am questioning both the premise behind the rule, and also the idea that the rule would be useful, in the unlikely event that the first premise is correct.

FWIW, I happen to think that the widespread use of simplistic metrics as if they measured something significant, and the elevation of rules-of-thumb to 'principles of design', are two of the reasons why so-called 'software engineering' has not really reached the status of true engineering. It's not just your rule that I take issue with!

1 ^ | v • Reply • Share ›

Voidy • 7 months ago

Nice non sequitur but no, having empty lines in the method body does not mean the method does more than one thing. Empty lines could separate LINQ code from C# code in a .NET project or they could delineate a particularly involved multi-line comment explaining why method is implemented the way it is.

On the other hand, your method could easily load a file, grep its contents and, I dunno, write the results to a relational database in ten lines or less and with no empty lines at all.

So if aggresively prohibiting usage of empty lines within method bodies works for you and your team, that's great, but urging everyone to follow suit is a bit of an overreach.

4 ^ | v • Reply • Share ›

Robert Watkins • 7 months ago

Actually, the worst thing about the sample code provided is not that it loads _and_ greps, but that it loads _then_ greps. It should filter as it reads - and, arguably, splitting the code up into the two methods hides that problem.

(Yes, i know it's a contrived example)

Smaller methods will reduce whitespace, and methods should limit how much they do. But banning empty lines entirely isn't going to solve the problem - just as it's easier to hit enter to add lines than to split a method up, it's even simpler not to add the lines at all. A better check, using Checkstyle, is to limit the number of executable statements per method, with a slightly higher limit on overall lines of code per method.

4 ^ | v • Reply • Share ›

Yegor Bugayenko author ➚ Robert Watkins • 7 months ago

We have that check in place too, in qulice. You're right about the load-then-grep problem, but it's just an example. Banning empty lines entirely is not the only restriction, of course, but it's a very important one. We're using it in over 20 projects (over 200K lines of Java code, all open source, I can give links), for the last two

years. The result is only positive.

• Reply • Share ›

Robert Watkins ➚ Yegor Bugayenko • 7 months ago

Well, if I look at the default configuration from qulice, I see that you allow methods that are 150 lines long, with a NPath complexity of 200, and a NCSS count of up to 50. Heck, you've even increased the fan-out complexity - allowing _more_ complex classes than the default permitted by CheckStyle.

I humbly suggest that if you have a method that is 150 lines long, with 50 executable statements, with a complexity count of up to 200, then the presence or absence of whitespace isn't the biggest issue. You're using whitespace as a proxy for another smell (method complexity), when you already have tools available to directly check method complexity.

I'd try lowering the NCSS count down to something like 15-20, and lower the fan-out complexity, to directly attack method & class complexity before I worried about something as trivial as whitespace.

2 • Reply • Share ›

Yegor Bugayenko author ➚ Robert Watkins • 7 months ago

We're using defaults from Checkstyle: https://github.com/teamed/quli... Never tried to tweak these numbers. Maybe we will. A method could be not complex enough to hit that NCSS or CC check, but with empty lines it will look just ugly.

• Reply • Share ›

daramasala • 7 months ago

Cool, never thought of that. I sometimes use empty lines to separate the 'init' part of the method from it logic, e.g. an empty line after instantiating the collection in your lines()

More from yegor256.com

yegor256.com recently published

How to Protect a Business Idea While Outsourcing

8 Comments Recommend

yegor256.com recently published

Constructors Must Be Code-Free

68 Comments Recommend

yegor256.com recently published

How Cookie-Based Authentication Works in the Takes Framework

11 Comments Recommend