

<http://www.yegor256.com/2014/11/03/empty-line-code-smell.html>

# An Empty Line is a Code Smell

3 November 2014 modified on 3 November 2014 Yegor Bugayenko

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<sup>↗</sup>. Why? In short, because a method should **not** contain "parts". A method should always do one thing<sup>↗</sup> and its functional decomposition should be done by language constructs (for example, new methods), and **never** by empty lines.

Look at this Java class (it does smell, doesn't it?):

```
final class TextFile {
    private final File file;
    TextFile(File src) {
        this.file = src;
    }
    public int grep(Pattern regex) throws IOException {
        Collection<String> lines = new LinkedList<>();
        try (BufferedReader reader = new BufferedReader(new FileReader(t
            while (true) {
                String line = reader.readLine();
                if (line == null) {
                    break;
                }
                lines.add(line);
            }
        }

        int total = 0;
        for (String line : lines) {
            if (regex.matcher(line).matches()) {
                ++total;
            }
        }
    }
}
```


```
    }  
    }  
    return total;  
}  
}
```

This method first loads the content of the file. Second, it counts how many lines match the regular expression provided. So why does method `grep` smell? Because it does two things instead of one — it loads and it greps.

If we make a rule, to avoid empty lines in method bodies, the method will have to be refactored in order to preserve the "separation of concerns" introduced by that empty line:

```
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 FileReader(t  
            while (true) {  
                String line = reader.readLine();  
                if (line == null) {  
                    break;  
                }  
                lines.add(line);  
            }  
        }  
    }  
}
```

```
    }  
    return lines;  
  }  
}
```



I believe it is obvious that this new class has methods that are much more cohesive and readable. Now every method is doing exactly one thing, and it's easy to understand which thing it is.

This idea about avoiding empty lines is also applicable to other languages, not just Java/C++/Ruby, etc. For example, this CSS code is definitely begging for refactoring:

```
.container {  
  width: 80%;  
  margin-left: auto;  
  margin-right: auto;  
  
  font-size: 2em;  
  font-weight: bold;  
}
```

The empty line here is telling us (screaming at us, actually) that this `.container` class is too complex and has to be decomposed into two classes:

```
.wide {  
  width: 80%;  
  margin-left: auto;  
  margin-right: auto;  
}  
.important {  
  font-size: 2em;  
  font-weight: bold;  
}
```

Unfortunately, using empty lines to separate blocks of code is a very common habit. Moreover, very often I see empty blocks of two or even three lines, which are all playing this evil role of a separator of concerns.

Needless to say, a properly designed class must have just a few public methods and a properly designed method must have up to ten instructions (according to Bob Martin). Empty lines inside methods encourage us to break this awesome rule and turn them into multi-page poems.

Of course, it's easier to just click `enter` a few times and continue to code right in the same method, instead of thinking and refactoring first. This laziness will eventually lead to code that is hardly maintainable at all.

To prevent this from happening in your projects, stop using empty lines inside methods, completely. Ideally, prohibit them in your automated build. In [qulice.com](http://qulice.com), a static analysis tool we're using in all Java projects, we created a custom Checkstyle check that prohibits empty lines in every method.