Design Patterns: Are They Dumb?

Written by Christian Weeks

Throughout the semester, we’ve been taught that the use of design patterns is the best way to organize our software. Moreover, we’ve been taught that they help make code less complex and easier to understand. Now for the sake of argument, I will take the stance that design patterns are stupid and do more harm than good.

Let’s examine why this seems to be the case for design patterns. Gerald Croles, a software architect, states that we should "never try to put design patterns in practice.” He also states that we should “never try to include them into the design of your new application.” He elaborates on this by explaining that design patterns, when used correctly, are great when put into practice. The issue that arises from this is that software engineers due to their schooling, believe that patterns aren’t a defined and immutable solution. Essentially, Croles is explaining that patterns aren’t an unchanging solution to whatever problem you're solving. Patterns shoud be used if and only if it fits the problem at hand. Without this in mind, you’ll make your code non-understandable, in which only the former developer understands the code, not the new person it’s been passed onto. Croles gives the advice to not see your current software patterns from a pattern perspective. Moreover, we should not seek to reconstruct our problem, so it fits the distinct pattern, or else you’ll end up writing mediocre solutions to problems you didn’t even have. Croles speaks from his own experience in the field, as he talks about how he doesn’t know the amount of times he’s reviewed lines of code littered with useless classes and pointless abstractions. With this being said, Croles gives the simplest solution, KISS, meaning Keep It Simple, Stupid. This is relatively self-explanatory, the most polished solution is the simplest one, the easiest to understand. Croles concludes the article by stating that you should develop for the current problem, not a hypothetical future problem. He continues by saying that developing for an unclear hypothetical future, can transform a two-day project into a two-month project. In an article from realfreemarket.org, the author states that design patterns might be useful if you’re on a massive project of 20+ people. They also discuss how if you use design patterns, the number of classes and methods in the project increases.

Now let’s look at a code example of a design pattern we covered in class, the composite pattern. We will compare the differences with, and without the pattern.

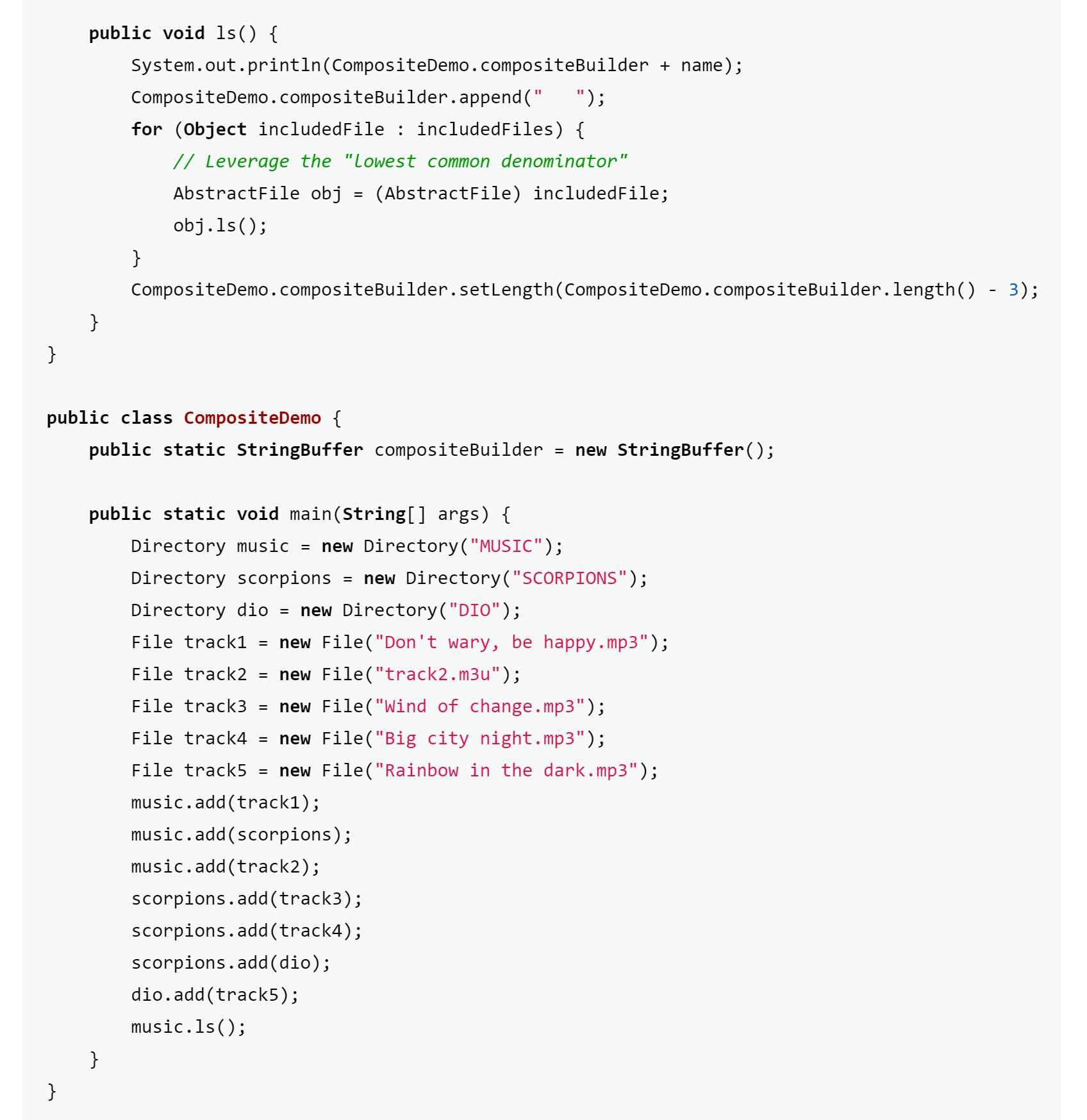
Before using composite in Java (courtesy of sourcemaking.com):





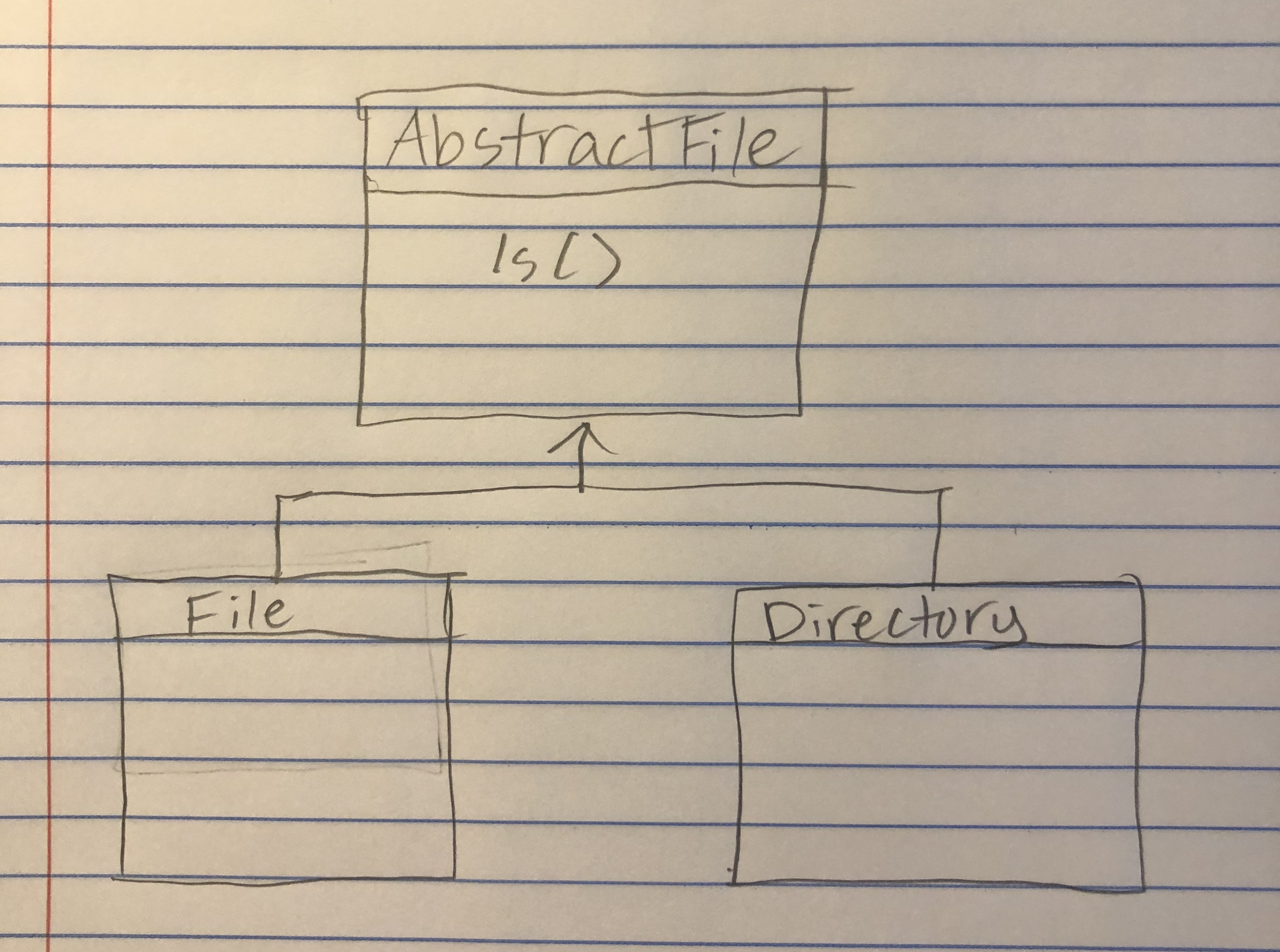
After using composite:





Now, looking at both side-to-side, there is one difference between them that is apparent at the beginning. Once the composite pattern is used, the interface AbstractFile is defined as a “lowest common denominator.” In the interface, there is one method called ls(), which each other class implements from the interface. The class File defines the ls() method, both before and after composite is used. With this interface, the class then defines the ls() method, by leveraging the lowest common denominator. Before the composite pattern is used, the method uses more comparisons, while with composite, the method makes a call to an enhanced for loop instead. Now, which one is more efficient, with or without composite. The answer to this question would be the use of the composite pattern. The composite is a structural design pattern that creates a tree structure of simple and composite objects. Without the composite pattern, this code segment doesn’t have a set structure of the objects. With the composite pattern, it helps streamline the structure of the code, such that there is one interface that each object uses, thus making a tree structure of the objects.

See below a figure of the structural tree of objects using the composite pattern:



Now you might be thinking what my stance is on design patterns, seeing as we looked at the benefits and drawbacks of them. Personally, I believe that design patterns can be very helpful, but can be stupidly implemented, which can make a project more complex than it needs to be. I think that I agree with Croles earlier, when he said that the best approach is to keep it simple. Design patterns have their place in code, don’t get me wrong, however they should be used sparingly, only if absolute necessary. I think simplicity is the best approach to software design, such that other programmers will be able to easily understand your code. I think the mix of simplicity and only if absolute necessary design pattern, is the most efficient way to develop software.

Sources:

<http://www.realfreemarket.org/blog/2012/06/21/design-patterns-suck/>

<https://sourcemaking.com/design_patterns/composite>

<https://medium.com/the-coding-matrix/https-medium-com-the-coding-matrix-dont-use-design-patterns-35bcff59dbb5>