During our discussions of design patterns over the last two weeks we’ve discussed many reasons why incorporating them into your software design can be advantageous. Briefly discuss one instance where using a design pattern may not be beneficial. Maximum points will be earned on this question for providing discussion presented during lecture and not on information obtained from the internet.

Ans: One time where using a design pattern might not be the best idea when the problem isn’t really complex or something that happens often. In those cases, the extra layers that come with a pattern can feel like too much.

In the SimUDuck example from the lecture, we saw this firsthand. Inheritance was used to share behavior, but that led to issues, like the RubberDuck having to override fly() and quack() just to stop them from doing what they normally would. It made things harder to manage instead of easier. We can say that design patterns are great when behavior changes and needs to be reused, but using them in simple cases can just clutter the design.

Like with the Strategy Pattern, if your system only has one or two behaviors that won’t really change, adding all that structure might actually make it worse. You’re creating extra classes and interfaces that aren’t doing much, and that just makes the code more confusing for someone else later on.

So overall, I think that patterns are tools not rules. If your design is already clean and isn’t going to change much, forcing a pattern into it might actually make it harder to understand and maintain. And like we saw in the slides, that goes against the KISS principle(Keep It Simple, Stupid.)