**Question Number 4 Answer:**

One instance where using a design pattern may not be beneficial is when the complexity added by the pattern outweighs its advantages—a key point raised during lecture. For example, the use of certain structural or behavioral patterns might promote flexibility, but this can come at the cost of making the design harder to understand and maintain. If the problem being solved is relatively simple or unlikely to change, introducing a pattern may introduce unnecessary abstraction and boilerplate code.

This was clearly demonstrated in our duck simulator example in class lecture, where adding behavior like fly() directly into the Duck superclass initially seemed like good reuse. But over engineering it with inheritance led to a maintenance headache when subclasses like RubberDuck and DecoyDuck had to override or disable behaviors that didn’t make sense for them. The takeaway was that sometimes applying patterns too early or unnecessarily complicates things, especially when the future need for flexibility isn't clear. So, while design patterns help avoid redesign, they can hurt clarity and increase complexity if applied in situations that don’t justify them.