DWA_07.4 Knowledge Check_DWA7

---

1. Which were the three best abstractions, and why?
    - A class or module should have only one reason to change(Repository pattern) - It separates the concerns of data access from other application logic, making the codebase more maintainable and testable.
    - Strategy pattern - This pattern allows you to define a family of interchangeable algorithms or behaviors and encapsulates each algorithm in a separate class. By abstracting the algorithms into separate classes, you can easily extend the system with new algorithms without modifying existing code.
    - Dependency injection(DI) - allows you to invert the control of creating and managing dependencies by providing them externally to a class rather than the class creating them itself. This promotes loose coupling between classes,

---

2. Which were the three worst abstractions, and why?

    - Lack of proper encapsulation - This violates the principle of loose coupling and can lead to breaking/fragile code that is tightly coupled to specific implementations. Feature Envy makes it difficult to change the internal implementation details without affecting dependent classes and can hinder the ability to maintain and test the codebase effectively.
    - Single responsibility principle– SRP can sometimes result in an increased number of classes or modules in a system which clashes with the concept should only have one reason to change. This means that a class should have only one job and do one thing.
    - The Liskov Substitution Principle-it is a very complex principle to use-requires adhering to a stricter set of requirements for subclassing and inheritance. This can limit the flexibility of designing and evolving the class hierarchy, as subclasses must fulfill certain expectations and maintain behavioral compatibility with the superclass.

---

3. How can The three worst abstractions be improved via SOLID principles.

1.Lack of proper encapsulation and feature envy - You can apply the Dependency Inversion Principle (DIP) and ensure that classes depend on abstractions rather than concrete implementations.
2.Single responsibility principle (SRP) - When applying SRP, focus on identifying cohesive responsibilities and ensuring that a class is responsible for a single, well-defined task. If a class becomes too large or has multiple responsibilities, you can refactor it into smaller classes, each with a specific responsibility.
