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 Inversion Principle-targeted towards loosely coupling software modules so that high-level modules (which provide complex logic) are easily reusable and unaffected by changes in low-level modules (which provide utility features).

---

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

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.

Open Closed Principle-

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.

Interface Segregation PrincipleWhen you implement this interface into any class, then the class needs to define all its methods, When any class implements this interface, all the methods must be defined even if you won't use them or if they don’t apply to that class.

---

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

The Interface Segregation Principle (ISP) states that “a client should never be forced to implement an interface that it doesn’t use, or clients shouldn’t be forced to depend on methods they do not use”. What does this mean?

you'll notice that if you change the payment gateway, you won't just need to add the class – you'll also need to make changes to the Store class. This does not just go against the Dependency Inversion Principle but also against the open-closed principle

---