Principles

Wednesday, 5 December 2018 12:12 AM

Separation of Concerns

- Reduces functional overlaps among code sections, limits ripple effect
- Class level, architectural level (n-tiers)
- Higher cohesion, lower coupling

Single Responsibility Principle

- A class should have one, and only one reason to change (vs God class)
- Separation of Concerns (SoC) applied at class level

Open-Closed Principle

- Module should be open for extension, closed for modification
    - Able to change behaviour without modifying code
- Usually requires separating specification (e.g. interface) from implementation

Liskov Substitution Principle

- Can substitute derived classes in place of base classes
    - Subclass should not be more restrictive than superclass (e.g. Cannot throw more exceptions)
- Not just a syntactical concept
    - E.g. BankAccount accepts up to $100, does not throw exception at $80. Child only accepts $10, error and break
    - E.g. Square.resize() vs Rectangle.resize()

Integration Segregation Principle

- Clients should not be forced to depend on methods it does not use

Dependency Inversion Principle

- High-level should not depend on low-level modules, both should depend on abstractions
- Abstractions should not depend on details (other way is alright)
- Does not reduce dependencies, rather changes their direction

Principles

- SOLID Principles
- Law of Demeter
    - Principle of least knowledge: don't talk to strangers
        - Minimises multi-level internal navigation, reduces coupling
    - Avoid a.getB().getC() unless a.getB() returns a new object

More concretely, a method `m` of an object `o` should invoke only the methods of the following kinds of objects:

- The object `o` itself
- Objects passed as parameters of `m`
- Objects created/instantiated in `m` (directly or indirectly)
- Objects from the direct association of `o`

- YAGNI: You aren't going to need it
- DRY: Don't repeat yourself (every piece of knowledge has a single and authoritative representation within system
- Brooks' Law: adding manpower to a late software project makes it later
    - Communication overhead of adding people