The L of SOLID

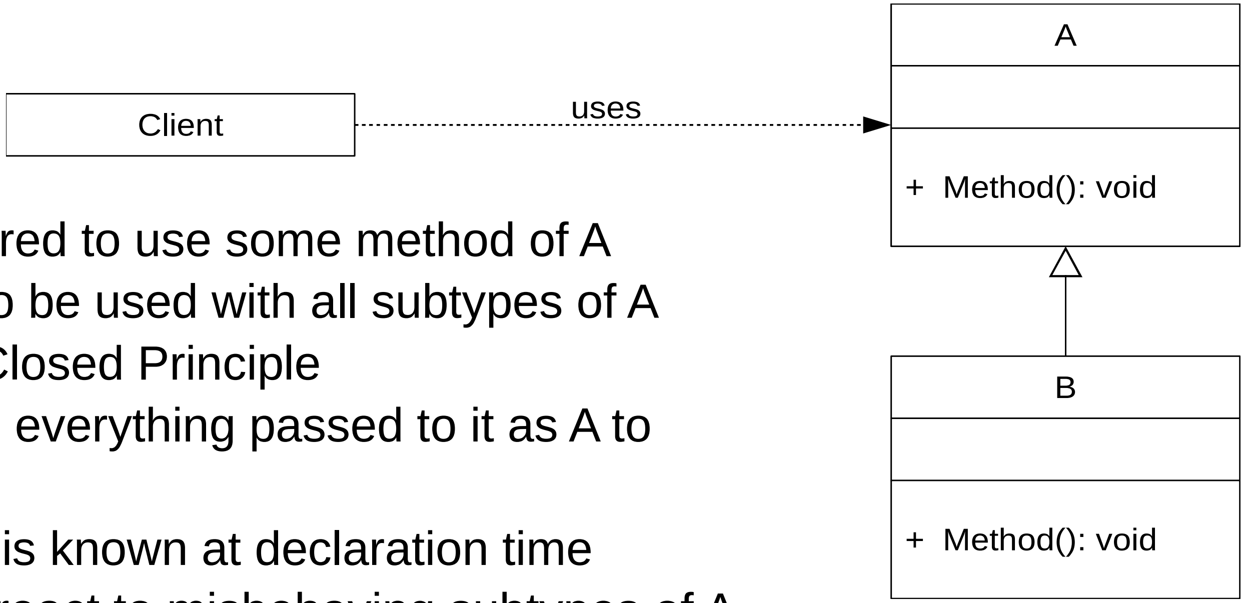
The Liskov Substitution Principle

Maximilian Meffert

The Liskov Substitution Principle

- Probably around since the late 1980’s
  - Liskov, Barbara. *Data Abstraction and Hierarchy*. SIGPLAN Notices, 23,5 (May 1988)
  - Barbara Liskov, Jeannette M. Wing: *A Behavioral Notion of Subtyping*. ACM Trans. Program. Lang. Syst. 16(6): 1811-1841 (1994)
- What does it say?
  - *“If for each object o1 of type S there is an object o2 of type T such that for all programs P defined in terms of T, the behavior of P is unchanged when o1 is substituted for o2, then S is a subtype of T.” (Barbara Liskov)*
  - *“Subtype Requirement: Let $\phi(x)$ be a property provable about objects x of type T. Then $\phi(y)$ should be true for objects y of type S where S is a subtype of T.”* (Barbara Liskov, Jeannette M. Wing)
  - *“Subtypes must be substitutable for their base types.”* (Robert C. Martin)
- What does it mean?
  - Subtypes have to retain behavior of its supertypes.
  - Subtypes have to comply with the same expectations its supertypes comply with.
  - Subtypes must not break expectations its supertypes comply with.

The Liskov Substitution Principle



- Client is declared to use some method of A
- Client can also be used with all subtypes of A
  - See Open/Closed Principle
- Client expects everything passed to it as A to behave like A
  - No subtype is known at declaration time
- Client cannot react to misbehaving subtypes of A without violating the Open/Closed Principle
  - E.g. with *instance-of* checks

The Liskov Substitution Principle

- Why is it a “good” thing to adhere to?
  - Change of requirements is immanent through the life cycle of most software
  - Decreases risk of regression because of hidden/indirect/behavioral coupling
  - Facilitates correct application of the Open/Closed Principle
  - Increases overall robustness

Modeling Inheritance

- Usually inheritance is taught as *Is-A* relationship used for modeling taxonomic hierarchies:
  - A circle is an ellipsis
  - A square is a rectangle
  - A set is a collection of elements
  - A pledging protection account is a giro account
- However, we tend to model *Is-A* relationships only considering syntactic traits or purposes:
  - Major/Minor axes
  - Right angles
  - *“[..] a gathering together into a whole of definite […] objects [...]”* (Georg Cantor)
  - System for managing money

Modeling Inheritance

- The problem is: entities in a *Is-A* relationship may behave differently when mutated (through inherited methods)
  - Circles: mutating axes
  - Squares: mutating width/height
  - Sets: adding elements
  - Pledging protection accounts: withdrawing money
- See Circle-Ellipse-Problem of inheritance in Object-Oriented Programming as case study of LSP violations

Modeling Inheritance

- Software is about behavior:
  - Inheritance imposes a semantic contract
  - Inheritance should be thought of and modeled as ***Behaves-Like*** relationship

Avoiding LSP violations

- Strategy 1: Design by Contract
  - Bertrand Meyer. 1995. Object-oriented software construction, New York: Prentice Hall.
  - {P} S {Q}: Every operation has pre- and postconditions (and invariants)
    - Preconditions cannot be strengthened in a subtype.
    - Postconditions cannot be weakened in a subtype.
    - (Invariants of the supertype must be preserved in a subtype)
  - Conditions should be documented:
    - Write Tests: Each subtype has to pass all tests of its supertype

Avoiding LSP violations

- Strategy 2: Avoid Inheritance
  - Only use interfaces for polymorphism
  - Interfaces only impose syntactic contracts which makes LSP violations more or less impossible

However,
avoiding something is the
best way to develop phobia

Examples

Thanks!

References

- Liskov, Barbara. *Data Abstraction and Hierarchy*. SIGPLAN Notices, 23,5 (May 1988)
- Barbara Liskov, Jeannette M. Wing: *A Behavioral Notion of Subtyping*. ACM Trans. Program. Lang. Syst. 16(6): 1811-1841 (1994)
- Bertrand Meyer. 1995. Object-oriented software construction, New York: Prentice Hall.
- Robert C. Martin. 2003. Agile Software Development, Principles, Patterns, and Practices, Prentice Hall.