

# 1 SOLID Design

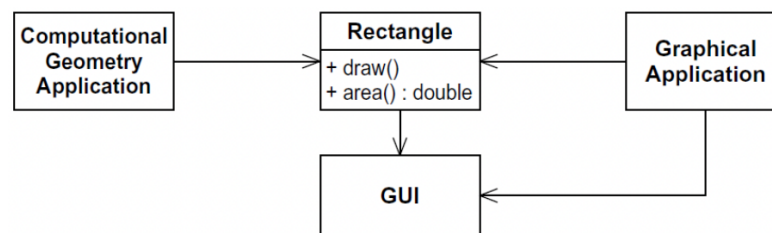
- What is SOLID?

- Single Responsibility Principle
- Open/Closed Principle
- Liskov Substitution Principle
- Interface Segregation Principle
- Dependency Inversion Principle

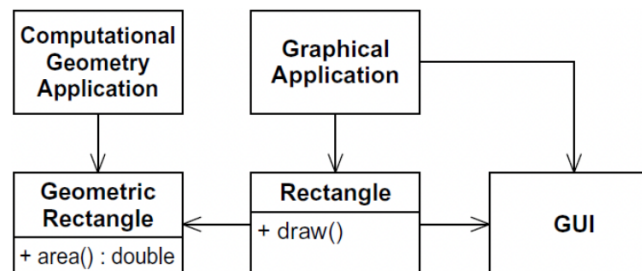
- Single Responsibility Principle (SRP)

*A class should have only one reason to change*

- If you can think of more than one motive for changing a class, then that class has more than one responsibility
- If a class has more than one responsibility, then the responsibilities become coupled
- Violating the SRP



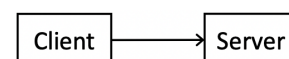
- Conforming to the SRP



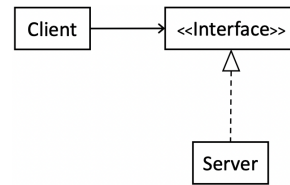
- The Open/Closed Principle (OCP)

*Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification.*

- When a single change to a program results in a cascade of changes to dependent modules, the design smells of rigidity.
  - \* If the Open/Closed principle is applied well, then further changes of that kind are achieved by adding new code, not by changing old code that already works.
- In Java, it is possible to create abstractions that are fixed and yet represent an unbounded group of possible behaviors
  - \* The abstractions are abstract base classes, and the unbounded group of possible behaviors is represented by all the possible derivative classes.
- Violating the OCP
  - \* Both classes are concrete
  - \* The **Client** uses the **Server** class



- Conforming to the OCP



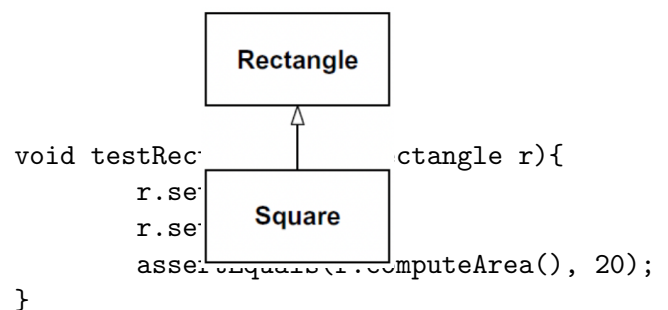
- The Liskov Substitution Principle (LSP)

*Subtypes must be substitutable for their base types.*

- Formally: 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  $T$ .
- Counter-example: “If it looks like a duck, quacks like a duck, but needs batteries – you probably have the wrong abstraction”
- Violating the LSP

Issues

- \* Inheriting **height** and **width**
- \* Overriding **setHeight** and **setWidth**
- \* Conflicting assumptions. For example:

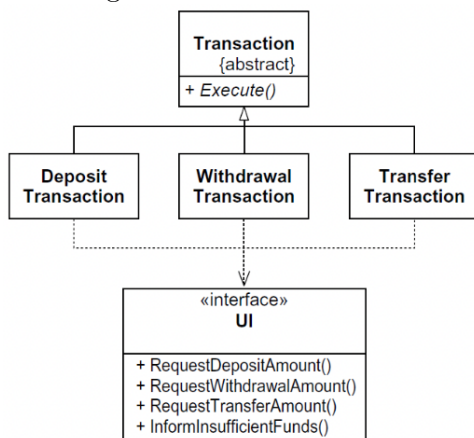


- Implication: A model, viewed in isolation, cannot be meaningfully validated.
  - \* The validity of a model can only be expressed in terms of its clients.
  - \* One must view the design in terms of the reasonable assumptions made by the users of that design.

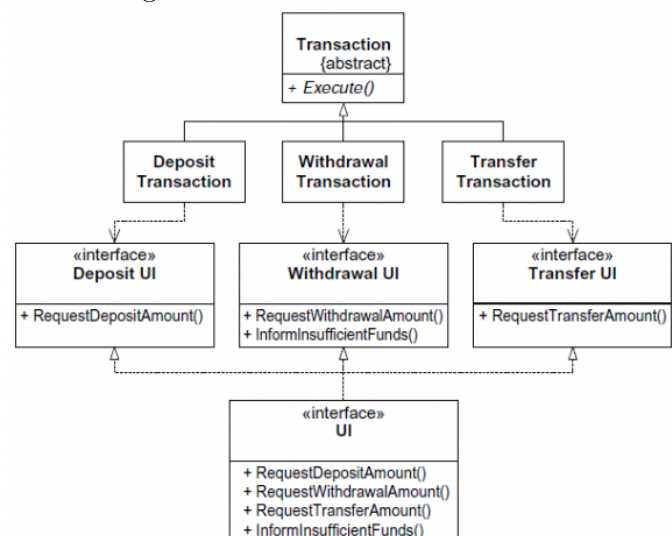
- The Interface Segregation Principle (ISP)

*Clients should not be forced to depend on methods that they do not use.*

- This principle deals with classes whose interfaces are not cohesive. That is, the interfaces of the class can be broken up into groups of methods where each group serves a different set of clients.
- When clients are forced to depend on methods that they don’t use, then those clients are subject to changes to those methods.
- Violating the ISP



- Conforming to the ISP



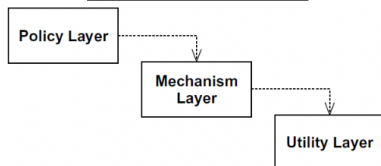
- The Dependency-Inversion Principle (DIP)

**A. High-level modules should not depend on low-level modules. Both should depend on abstractions.**

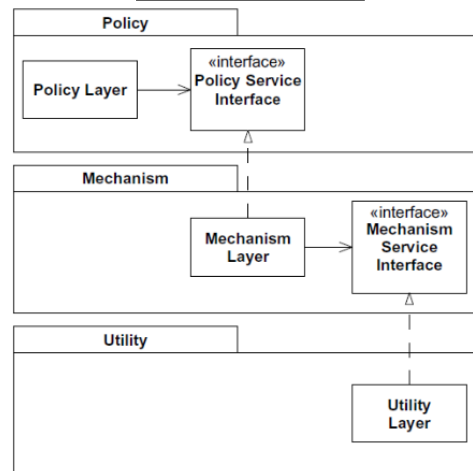
**B. Abstractions should not depend on details. Details should depend on abstractions.**

- The modules that contain the high-level business rules should take precedence over, and be independent of, the modules that contain the implementation details.
- When high-level modules depend on low-level modules, it becomes very difficult to reuse those high-level modules in different contexts.

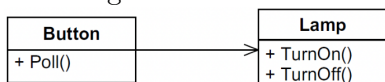
### Naïve Layering



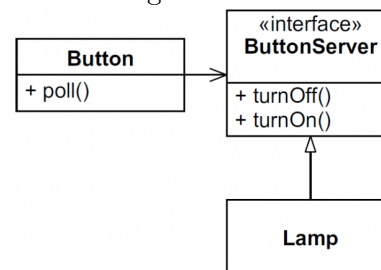
### Inverted Layers



- Violating the DIP



- Conforming to the DIP



- Design Smells

- Symptoms of poor design
- Often caused by the violation of one or more of the design principles
  - \* For example, the smell of *Rigidity* is often a result of insufficient attention to OCP.
- These symptoms include:
  1. Rigidity – The design is hard to change.
  2. Fragility – The design is easy to break.
  3. Immobility – The design is hard to reuse.
  4. Viscosity – It is hard to do the right thing.
  5. Needless Complexity – Overdesign.
  6. Needless Repetition – Mouse abuse.
  7. Opacity – Disorganized expression.