SOLID

SRP (Single Responsibility Principle)

SRP

A class should have only one reason to change.

➔ Meaning:
- A class should have one responsibility.
- A class should be changed based on the request/change by only one actor.

➔ Benefits:
- Smaller class that is easy read, maintain and unit test.

SRP (Single Responsibility Principle)

- God class:
  - A class that does or know too many things
  - Opposite is SRP

- Real life God object

- Real life SRP object

SRP (Single Responsibility Principle)

- Violation of SRP
  - We must include GUI component (.class file) to draw. Even though **ComputationalGeometryApplication** do not use it.
  - If **GraphicalApplication** causes the **Rectangle** to change for some reason, that change may force rebuild, retest, and redeploy **CGA**.

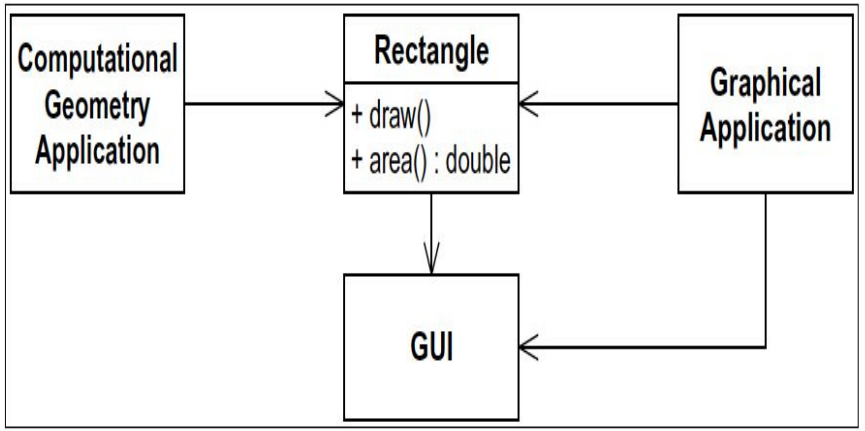


Figure 8-1 More than one responsibility

SRP (Single Responsibility Principle)

- Better Design of SRP
  - Now changes made to the way rectangles are rendered cannot affect the **ComputationalGeometryApplication**.

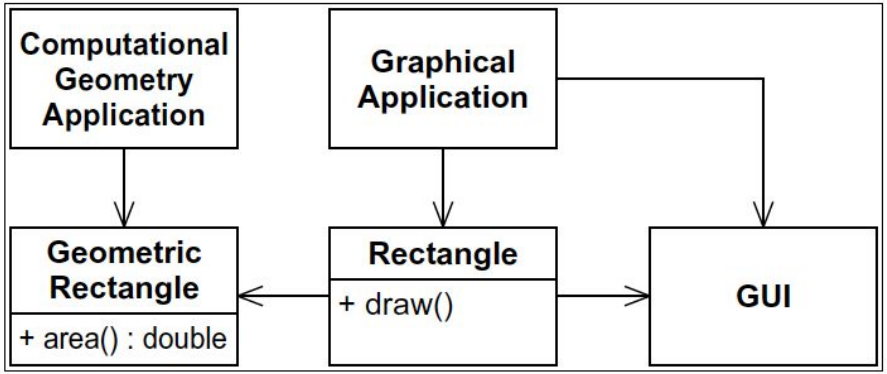


Figure 8-2 Separated Responsibilities

SRP (Single Responsibility Principle)

- BankService
  - deposit(amount, accountNumber)
  - withdraw(amount, accountNumber)
  - printDetails(accountNumber)
  - getLoanInfo(loanType)
  - sendOTP(medium)

- BankService
  - deposit(amount, accountNumber)
  - withdraw(amount, accountNumber)
- PrintService
  - printDetails(accountNumber)
- LoanService
  - getLoanInfo(loanType)
- NotificationService
  - sendOTP(medium)

SRP (Single Responsibility Principle)

- TextManipulator
  - text
  - getText()
  - appendText(text)
  - findAndReplace(text)
  - findAndDelete(text)
  - printText()
  - printWordPosition(word)
  - printRangeOfCharacter(start, end)

- TextManipulator
  - text
  - getText()
  - appendText(text)
  - findAndReplace(text)
  - findAndDelete(text)
- TextPrinter
  - TextManipulator
  - printText()
  - printWordPosition(word)
  - printRangeOfCharacter(start, end)

SRP Violation Smells

- Class has too many instance variables and methods
  - Especially, a group of them appeared together, changed together
- The test class becomes too complicated
  - Too many test cases
  - Too many mock objects
- Use of **and**, **or**, in names
- Class with low cohesion
  - Many parts of a class not related to each other
- Class / Method is long

SRP Violation Smells

- Unable to encapsulate of Module
- Class has too many dependencies
- Rigidity
  - Difficult to make even simplest change.
  - Cascade of subsequent changes in dependant modules.
  - Makes it difficult to do reasonable estimate - “It was a lot more complicated than I thought”
- Fragility:
  - Break the program in many places when change is made in one place.
  - Problem arises even in conceptually unrelated area.
  - Fixing new problems creates more issue.

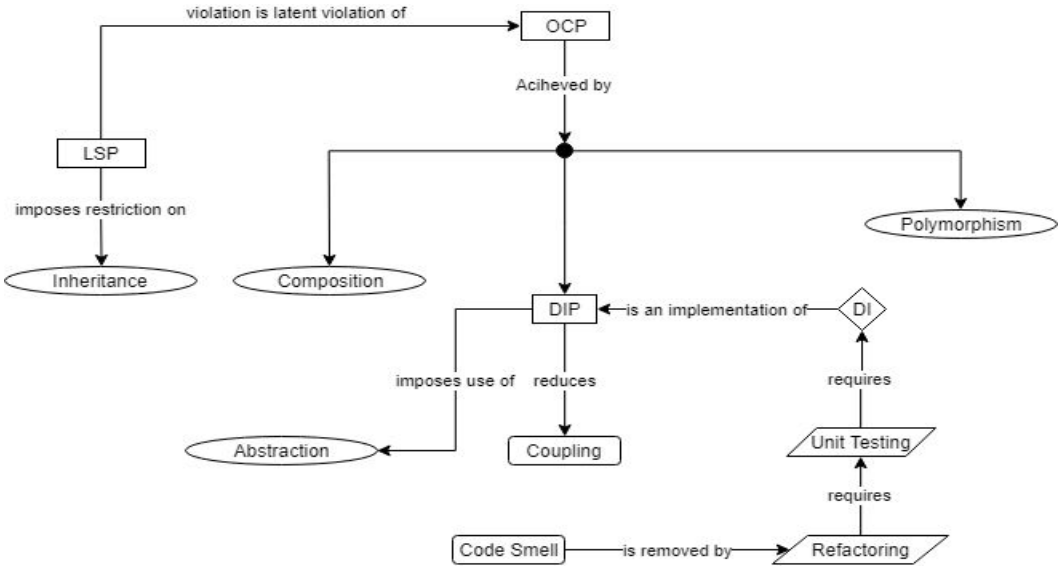
Interrelationship among OO concepts

- SRP
  - SRP ensures highly cohesion of class.
  - ISP also deals with cohesion, but cohesion of interface, not class.
  - SRP violation results in large class.
- OCP
  - OCP is achieved by a combination of composition, polymorphism and DIP.
  - Overthinking of OCP may result in several code and design smells. Examples - Speculative generality, needless complexity.
  - OCP violation results in conditional statements, rigidity.
- LSP
  - LSP imposes rules on inheritance.
  - Violation of LSP is also a latent violation of OCP.
  - Refused bequest is a special case of violation of LSP.

Interrelationship among OO concepts

- ISP
  - ISP ensures cohesion of interface.
- DIP
  - DIP imposes use of abstraction.
  - DIP reduces coupling.
  - DIP has several implementation, DI (Dependency Injection) is one of them.
  - Violation of DIP causes fragility,
- Abstraction
  - Many code smells can be simply identified as missing abstraction. Examples of such code smells -
    - long method - method could be extracted, method is a form of abstraction
    - long class - extract class, class is a form of abstraction
    - conditional complexity - extract method, method is a form of abstraction.
    - data clump - extract clumps of data to class, class is a form of abstraction.

Interrelationships between OO concepts



1. OCP is the fundamental principle. This is because it is related to **change** in software.
    a. DIP helps implementing OCP
    b. LSP is a subset of OCP
2. Encapsulation is still missing in the graph