Software Design
9. Object-oriented Design

Natsuko Noda
nnoda@shibaura-it.ac.jp

ソフトウェア設計論
9. オブジェクト指向設計

野田 夏子
nnoda@shibaura-it.ac.jp

Today's Topics

- Data oriented approach
- Object-oriented design

本日のお題

- データ中心アプローチ
- オブジェクト指向設計

Data oriented approach

データ中心アプローチ

cf. SE-J, 5.4

Basic viewpoints of design (Review)

- How can we find and design modules to realize good modularization?
  - There is no unique answer. However, there are basic viewpoints of design and we choose and/or combine these viewpoints.

Viewpoints	Typical methodologies	Typical diagrams & models
Function	Structured method Process oriented approach	Data flow diagram Structured chart
Information, data	Data oriented approach	Entity relationship diagram Relational model
State	State transition design	State machine diagram Sequence diagram

設計の基本的な視点 (復習)

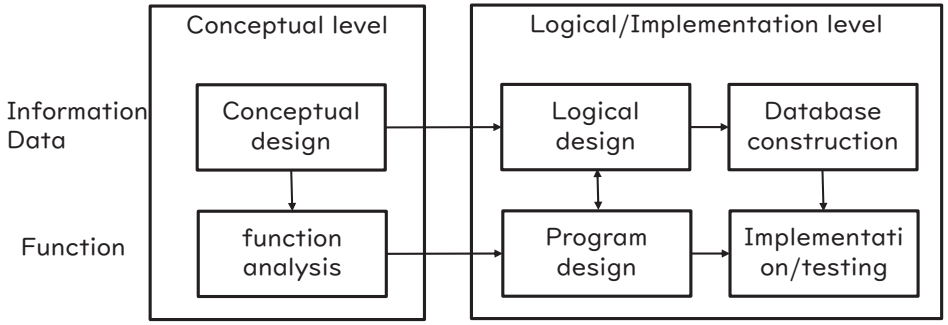
- どのようにして良いモジュール化を実現?
  - 唯一の答えはない。しかし、設計の基本的な視点は存在し、それらから選んだり、組み合わせたりして、設計を行う

視点	典型的な手法	典型的な図やモデル
機能	構造化手法 プロセス中心アプローチ	データフロー図 ストラクチャチャート
情報, データ	データ中心アプローチ	実体関連図 リレーショナルモデル
状態	状態遷移設計	ステートマシン図 シーケンス図

Data oriented approach

- The structure of information and data is clarified before the design of functionality.
- The structure of information and data
  → database design

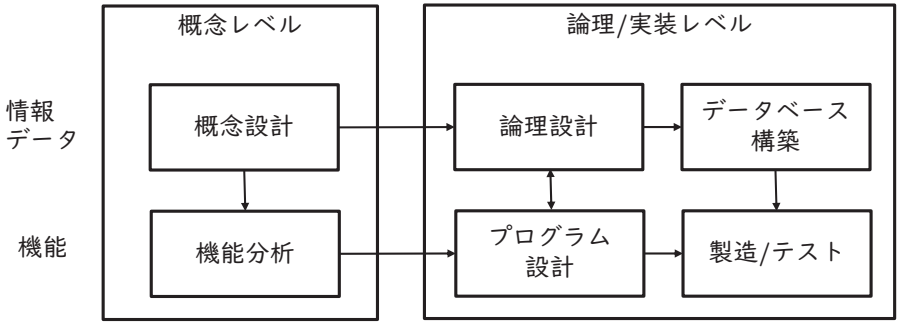
Example of data oriented approach



データ中心アプローチ

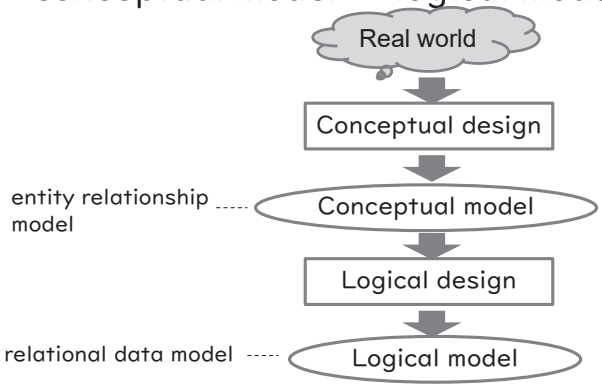
- 機能の設計をする前に、情報やデータの構造の明確化を行うアプローチ
- 情報やデータの構造
  → データベース設計

データ中心アプローチの例



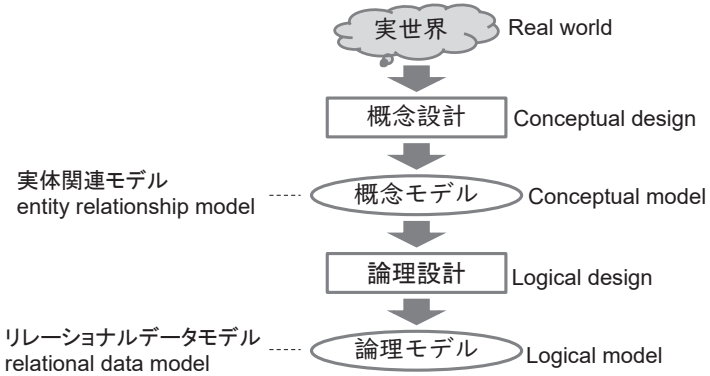
Process of data modeling

- conceptual design
  - Real world → conceptual model
- logical design
  - conceptual model → logical model



データモデリングの流れ

- 概念設計
  - 実世界 → 概念モデル
- 論理設計
  - 概念モデル → 論理モデル



Entity relationship model

- Basic concepts
  - Entity: Description of a thing or an object existing in the real world
  - Relationship: a model element specifying a relationship between entities

実体関連モデル

- 基本概念
  - 実体：実世界をモデル化しようとする際に、その存在を認識できる対象を包括的に述べたもの。実世界の様々な「もの」や「こと」
    関連：実体同士の相互関係をモデル化したもの

Entity relationship diagram

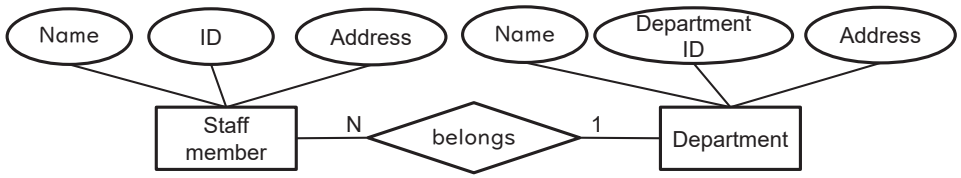
- Graphical expression of entity relationship model
  - There are various notations
- Notations of basic elements
  - Entity set : rectangle
  - Relationship set : diamond shape
  - Attribute : ellipse

実体関連図

- 実体関連モデルをグラフィカルに表現したもの
  - 様々な記法がある
- 基本要素の図形表現
  - 実体集合：長方形
  - 関連集合：ひし形
  - 属性：だ円

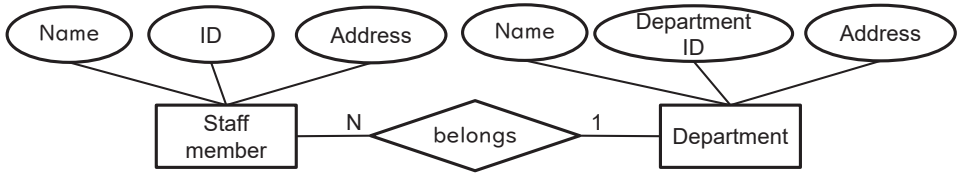
Entity relationship diagram

- Example



実体関連図

- 例



Relational data model

- Introduced by
  - Edger F. Codd (1970)
- Basic element
  - Relation
    - As intuitive understanding, two dimensional table

Name	Cacao percentage
AAA	18%
BBB	23%
CCC	36%

リレーショナルデータモデル

- 提案者
  - Edger F. Codd (1970年)
- 基本要素
  - リレーション
    - 直観的には2次元の表

名称	カカオ含有量
コアラの街	18%
たけのこの山	23%
大枝	36%

Process of data modeling

- conceptual design
  - Real world → conceptual model
  - To analyze information and events of real world, and to model them as entity relationship diagram
  - Careful consideration is needed.
- logical design
  - conceptual model → logical model
  - To transform entity relationship diagram to relation model
  - There are transformation rules we can use.

データモデリングの流れ

- 概念設計
  - 実世界 → 概念モデル
  - 実世界の情報、出来事をよく観察・分析し、実体関連図に表現
  - 熟考が必要
- 論理設計
  - 概念モデル → 論理モデル
  - 実体関連図を論理モデルに変換
  - 変換ルールが存在

Object-oriented Design

オブジェクト指向設計

cf. SE-J, 5.6

Object-oriented design

- Around 1980- : object-oriented programming languages
- Around 1990- : object-oriented design
- Advanced method based on the basic viewpoints of design.

オブジェクト指向設計

- 1980年頃から：オブジェクト指向プログラミング言語の登場
- 1990年頃から：オブジェクト指向設計の登場
- (先に述べた)設計の基本的な視点に基づく発展的な手法

Characteristics of object-oriented design

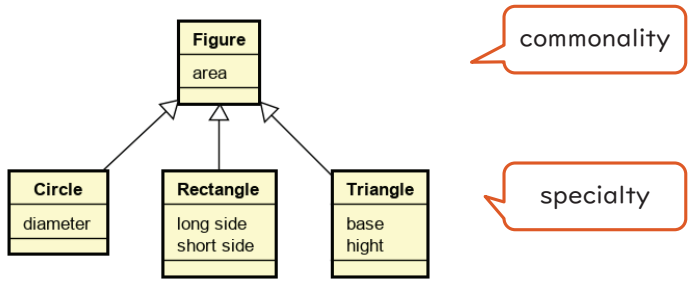
- Object-oriented design is the design method in which a class and an object is a unit of modularization.
- Characteristics
  - Inheritance
  - Polymorphism
  - Delegation

オブジェクト指向設計の特徴

- クラスとオブジェクトをモジュール化の単位に用いる設計手法
- 特徴
  - 継承 (Inheritance)
  - ポリモルフィズム (Polymorphism)
  - 委譲 (Delegation)

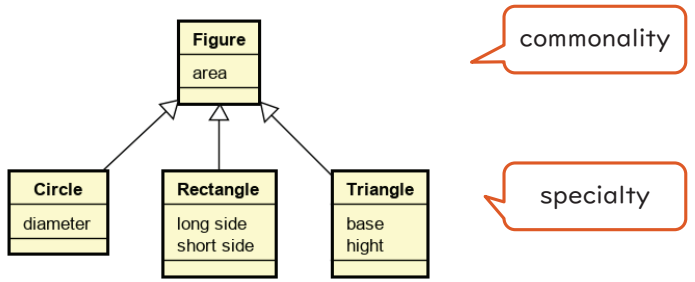
Inheritance

- Good for independence of objects.
  - Able to localize effects of changes.



継承

- オブジェクトの独立性のために有効
  - 変更の影響を局所化できる



Polymorphism

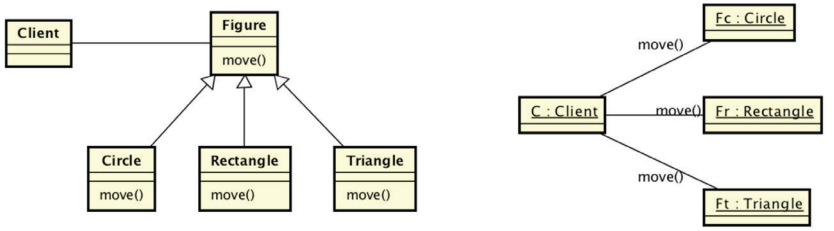
- Different classes may have the same operation name.
  - The specification of the operation is identical for each class; classes can implement the operation differently.
  - Objects with identical interfaces to be substituted for each other at run-time.
- Useful for achieving a highly independent module structure

ポリモルフィズム

- 異なるクラスが同じ操作名を持つことができる
  - 操作の仕様はそれぞれのクラスで同じであるが、その実装は異なる
  - 同じインタフェースのオブジェクトが実行時に互いに置換される
  - メソッドが呼ばれた時に実際に動作する実装が動的に変わる
- 独立性の高いモジュール構造の実現に有用

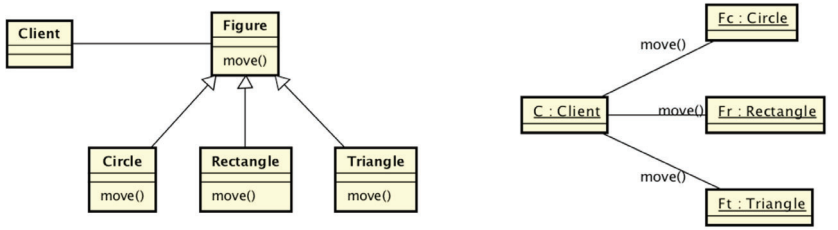
Polymorphism (Cont.)

- Example.
  - Client has to know that each subclass of Figure has a method named "move()".
  - At run-time, the appropriate implementation of move() is executed.
  - If Pentagon is added as a subclass of Figure, no change to client is needed.



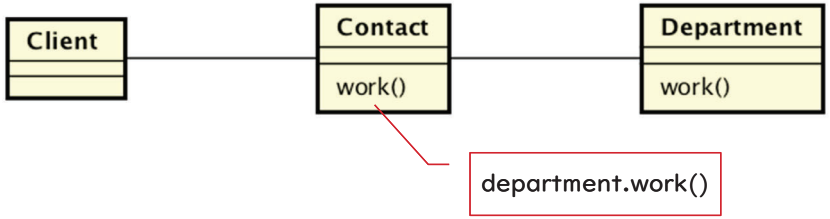
ポリモルフィズム (Cont.)

- 例：
  - ClientはFigureのサブクラスがメソッドmove()を持つことを知っていれば良い
  - 実行時は適切なmove()の実装が実行される
  - Pentagon(五角形)がFigureのサブクラスとして追加されてもClientの変更は必要なし



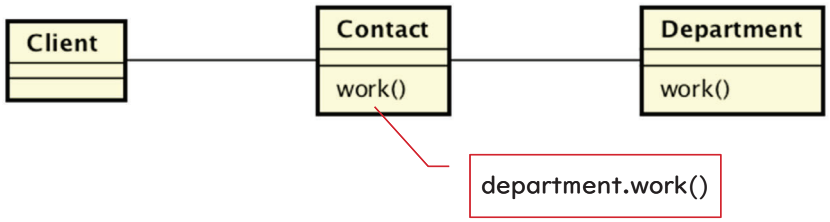
Delegation

- A class passes a request received to other classes and let them to handle it.
  - Concentration only on own responsibilities.



委譲

- 受け取った要求の処理を他のクラスに任せる
  - 他に対して何らの動作や責務を割り当てる動作
  - 自身の責任に集中



How to find appropriate classes?

- In object-oriented design, a class is the unit of modularization.
- Focus on the role that classes play in the system.
- Methods
  - CRC
  - Design by Contract

クラスの発見

- オブジェクト指向設計では、クラスがモジュール化の単位
- クラスがシステムの中で果たす役割に注目
- 方法
  - CRC
  - 契約による設計

CRC

- Class-Responsibility-Collaborator
- A method for examining class roles using CRC cards
  - CRC card：has three areas; class name, responsibilities (methods)、Collaborators (another classes that a class interacts with to fulfill its responsibilities)
  - Use the CRC card to discuss the classes you need.

Class Name	
Responsibilities	Collaborators

Ex.

Student	
Register for a class Enroll in a Seminar Request transcripts	Seminar

CRC

- Class-Responsibility-Collaborator
- CRCカードを利用したクラスの役割の検討手法
  - CRCカード：クラス名、責務(メソッド群)、協調(その責を果たすために必要なクラス群)を記述
  - CRCカードを利用して、必要なクラスを議論

クラス名	
責務	協調

例

学生	
授業登録 ゼミ登録 成績証明書請求	ゼミ

Design by Contract (DBC)

- A method of describing the interface of a class in a specification called a contract.
  - Introduced by B. Meyer
- Contract
  - The conditions that must be met between the class that provides the method and the class that uses the method.
  - Preconditions: conditions that must be satisfied before calling a method.
  - Postconditions: conditions that should be satisfied after calling a method.
- Contract and class responsibility
  - Caller of the method must guarantee preconditions.
  - Provider of the method must guarantee postconditions.

契約による設計 (DBC)

- クラスのインタフェースを契約と呼ばれる仕様で記述する方法
  - B. Meyerが提案
- 契約
  - メソッドを提供するクラスと、そのメソッドを利用するクラスとの間で成立すべき条件
  - 事前条件：メソッドを呼ぶ前に成立すべき条件
  - 事後条件：メソッドを呼んだ後に成立すべき条件
- 契約とクラスの責任
  - 事前条件は、メソッドを呼び出す側が保証する責任がある
  - 事後条件は、メソッドを提供する側が保証する責任がある

Design by Contract (DBC) (Cont.)

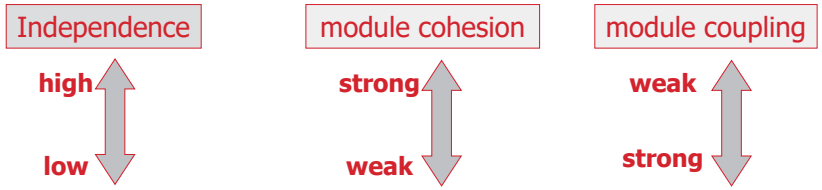
- Ex： Class with a "get_quotient" method
  - Method parameters: divident, divisor
  - Method return value: quotient
  - Precondition: divisor is not zero.
  - Postcondition: divident = quotient × divisor

契約による設計 (DBC) (Cont.)

- 例：「商を返す」メソッドを持つクラス
  - メソッドの引数：被除数、除数
  - メソッドの返り値：商
  - 事前条件：除数が0でない
  - 事後条件：被除数 = 商×除数

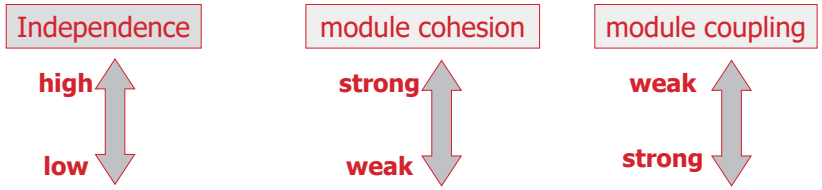
Module cohesion and coupling (Review)

- Module cohesion: Strength of ties within a module.
  - The role that the module plays in a system is:
    clear → cohesion is strong.
    vague → cohesion is weak.
- Module coupling: Ties between modules.
  Relationships between modules.
  - Only necessary and sufficient relations exist.
    → coupling is weak.
    Unnecessary relations exist.
    → coupling is strong.



モジュール強度・モジュール結合度 (復習)

- モジュール強度: モジュール内の結びつきの強さ
  - システムがモジュールで果たす役割が
    明確 → 強度が強い
    あいまい → 強度が弱い
- モジュール結合度: モジュール間の結びつきの強さ、モジュール間の関係性
  - 必要かつ十分な関係しかない
    → 結合度が弱い.
    不必要な関係が存在する
    → 結合度が強い



Design guideline

- Principles for achieving the desired module structure
  - Single Responsibility Principle (SRP)
  - Open Closed Principle (OCP)
  - Liskov Substitution Principle
  - Dependency Inversion Principle (DIP)
  - Interface Segregation Principle (ISP)

設計のガイドライン

- 望ましいモジュール構造を実現するための原則
  - 単一責任の原則 (SRP)
  - 開放閉鎖原則 (OCP)
  - リスコフの置換原則 (LSP)
  - 依存関係逆転の原則 (DIP)
  - インタフェース分離の原則 (ISP)

Single Responsibility Principle (SRP)

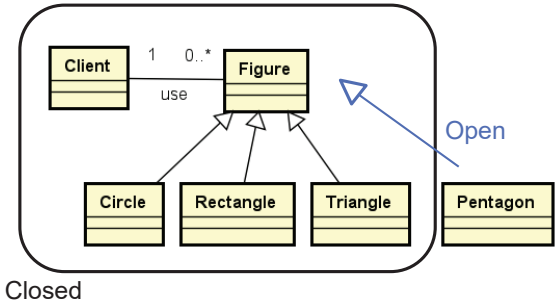
- A class should have only one reason to change.
  - Changing a class is necessary when the role the class plays changes.
  - One class should not have more than one role.
    - Multiple roles → Modules become tightly coupled.

単一責任の原則 (SRP)

- クラスは変更する理由をひとつだけ持つべきである
  - クラスを変更することが必要なのは、クラスが果たす役割が変わる場合
  - 1つのクラスに複数の役割を持たせてはいけない
    - 複数の役割を持つ → モジュールが強結合になる

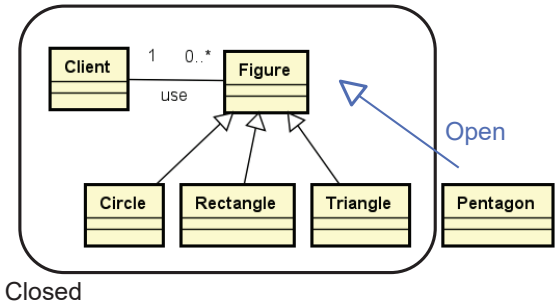
Open Closed Principle (OCP)

- class should be able to extend its behavior without modifying it.
  - "Open" for extension. (Extension is easy.)
  - "Closed" for modification. (Modification is localized.)



開放閉鎖原則 (OCP)

- クラスを修正することなくその振舞いを拡張できるべきである
  - 拡張に対して「オープン」(拡張が容易)
  - 修正に対して「クローズ」(修正が局所化される)



Liskov Substitution Principle (LSP)

- The derived class must be replaceable with the original class.
  - A subclass "is a" parent class.
  - A Subclass must behave as its parent classe.

  - Note: Liskov is a person name.

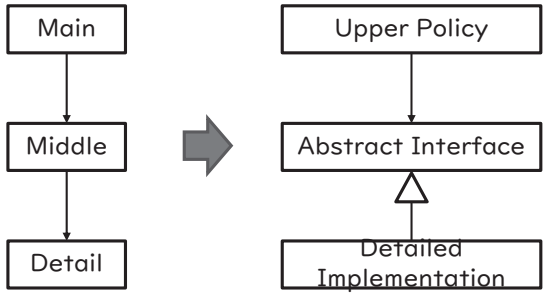
リスコフの置換原則 (LSP)

- 派生するクラスは、元となったクラスと置き換え可能でなければならない
  - サブクラスは、親クラスでなければならない
  - サブクラスは、親クラスとして振舞えなければならない

  - 注：リスコフは人名

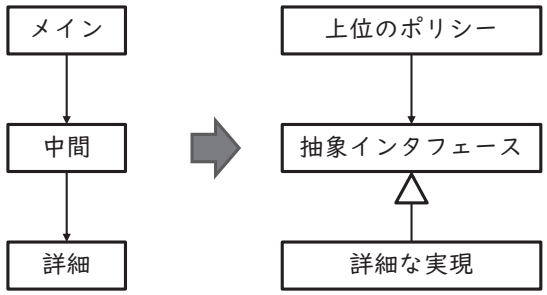
Dependency Inversion Principle (DIP)

- Should not depend on concrete classes (implementation), but on the abstractions (interface).
  - Increase independence through intervening interfaces, rather than relying directly on lower layers.



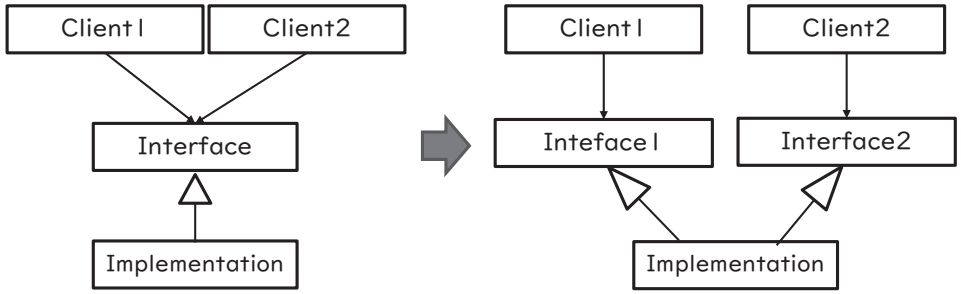
依存関係逆転の原則 (DIP)

- 具体(実装)に依存するのではなく、抽象(インタフェース)に依存するべきである
  - 下位レイヤに直接依存するのではなく、インタフェースを介在させて独立性を高める



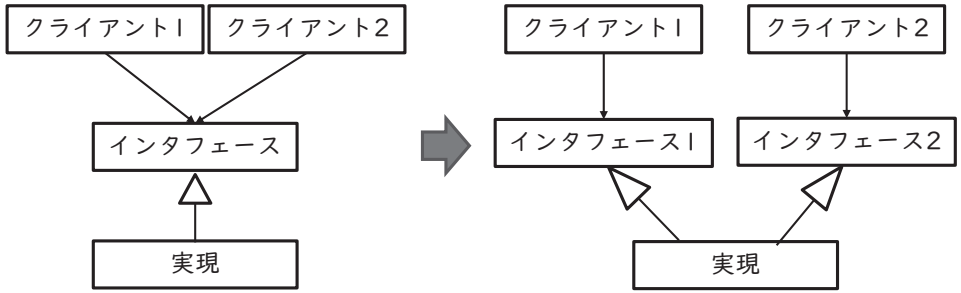
Interface Segregation Principle (ISP)

- A fine-grained interface should be provided for each client.
  - One interface for a set of methods needed by different clients exposes unnecessary methods and introduces the risk of unnecessary coupling.



インタフェース分離の原則 (ISP)

- クライアント毎にきめ細かいインタフェースを作るべきである
  - 異なるクライアントが必要とするメソッド群をひとつのインタフェースにすると、不要なメソッドが公開され、不必要な結合をもたらす危険がある



OO design and UML

- Before designing, develop a use case diagram (and use case descriptions) for requirements definition.
- To consider data structure, we may use class diagrams instead of entity relationship diagrams.
- Using CRC cards and/or other methods, find class candidates. Based on these candidates, we design static structure of the software using class diagrams.

オブジェクト指向設計とUML

- 設計の前に、要件定義のためのユースケース図(およびユースケースの記述)を作成する。
- データ構造の検討のために、実体関連図ではなく、クラス図を使用することもある
- CRCカード等を用いて、クラスの候補を見つける。その候補をもとに、クラス図を用いてソフトウェアの静的構造を設計する

OO design and UML (Cont.)

- To consider collaborations between classes in developing class diagrams, we may use sequence diagrams.
- To design the behavior of the class, we may use state machine diagrams. Especially for embedded software development, we utilize state machine diagrams.

オブジェクト指向設計とUML (Cont.)

- クラス図作成の際、クラス間の協調動作を検討するためには、シーケンス図を使用することができる
- クラスの振舞いの設計するには、ステートマシン図を用いることができる。特に組込みソフトウェアの開発では、よくステートマシン図が活用される

For your review and more

復習と発展学習

Designing state changes

- Draw a state machine diagram of an automatic ticket gate.
- Develop a state transition table corresponding to the above developed state machine diagram.

状態遷移設計

- 自動券売機のステートマシン図を描いてみよう
- 上記のステートマシン図に対応する状態遷移表を作ろう

CRC

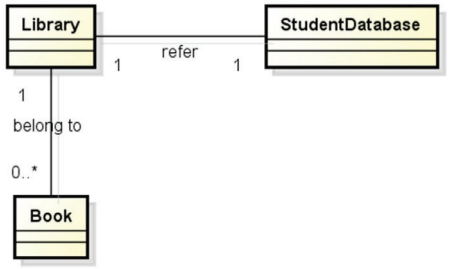
- Develop a CRC card of class "Contact" on p.37

CRC

- P.38のクラス「Contact」のCRCカードを書いてみよう

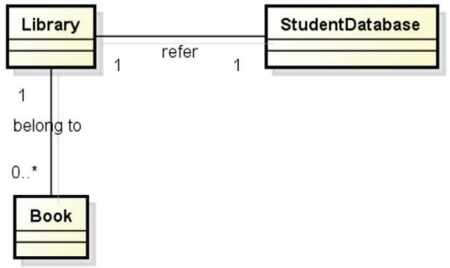
Is this model appropriate?

- University library management



このモデル化は適切か？

- 大学図書館管理システムのクラス図



Survey on design principles

- Study more deeply on design principles.
- Reference
  - http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod
    - "class design" on the above page:
      SRP, OCP, LSP, ISP, DIP
  - Agile Software Development – Principles, Patterns, and Practices
    by Robert C. Martin

設計原則

- 設計原則についてさらに調べてみよう
- 参考文献
  - http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod
    - 上記ページの"class design":
      SRP, OCP, LSP, ISP, DIP
  - Agile Software Development – Principles, Patterns, and Practices
    by Robert C. Martin
  - (邦訳)　「アジャイルソフトウェア開発の奥義 第2版」
    瀬谷啓介　訳
    - 邦訳は例がJavaで記述されているので、C++を知らない人はこちらの書籍の方がわかりやすいかも