ソフトウェア工学 第7回
― モジュール設計 ―

大連理工大学・立命館大学 国際情報ソフトウェア学部

大森 隆行

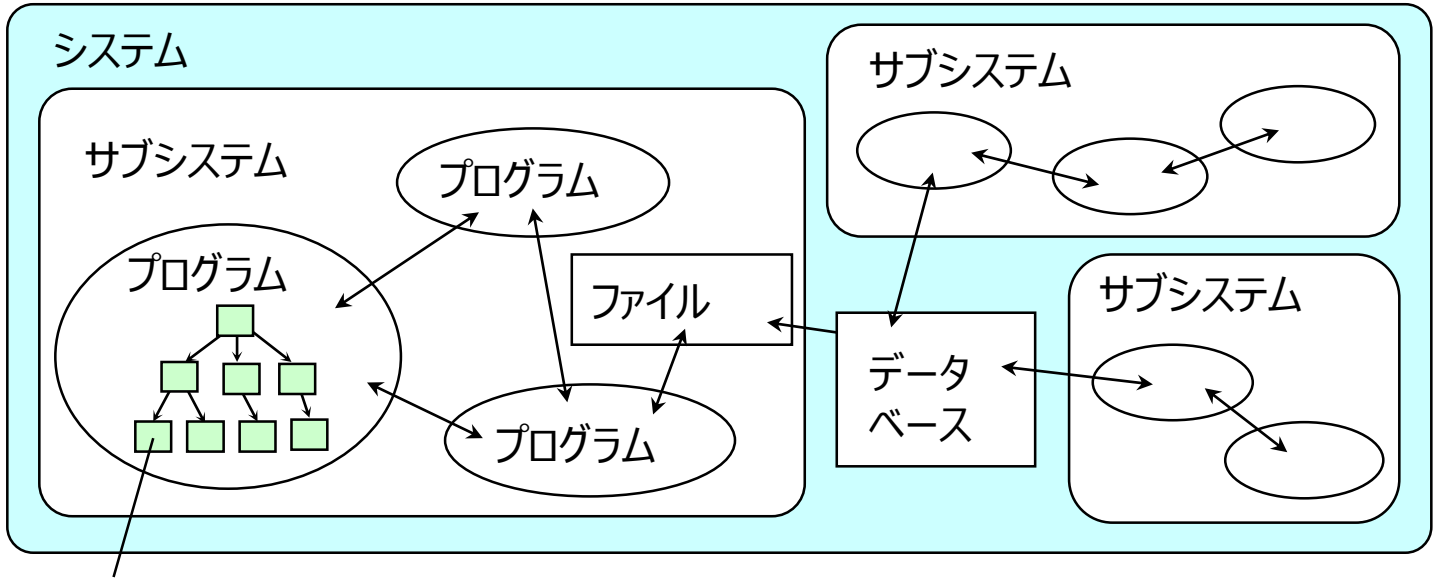
講義内容

- モジュール設計
  - 概説
  - モジュール分割の評価基準
  - モジュール分割技法
  - オブジェクト指向におけるモジュール設計(参考)

モジュール設計

- ソフトウェアをモジュールに分割し構造化する作業
  - ソフトウェア ＝ 複数のモジュールで構成
- モジュール(module)
  - 一連の機能をひとまとめにしたプログラムの構成単位
  - サブルーチン(subroutine)、関数、手続き、クラス等
- モジュール分割の利点
  - 抽象化：詳細を把握せずとも利用可能
  - 開発効率：並行して開発可能
  - 再利用：既存のモジュールを再利用可能
  - 変更容易性：変更範囲を局所化

モジュール



モジュール

- 複数の文で構成され、独立して識別可能な名前をもつ
- 決められたインタフェース(interface)を通してのみ呼出可能である

(例)　　決められた関数名・引数で呼び出し

```
void func(int x){...}   <-   func(123);
```

実際にモジュールという言葉が指すものは言語等により異なる

講義内容

- モジュール設計
  - 概説
  - モジュール分割の評価基準
  - モジュール分割技法
  - オブジェクト指向におけるモジュール設計(参考)

モジュール分割の評価基準

- 分割の観点
  - モジュールの大きさ
    - 例：モジュールを構成する文の数
  - モジュールの簡潔さ
    - 簡潔 ⇔ 単一の機能のみ含む ⇔ 汎用化しやすい
- 独立性の観点
  - 機能独立性
    1つの目的に沿った機能のみ提供し、他のモジュールとの相互作用が少ない
  - 情報隠蔽（カプセル化）
    モジュールのインタフェースと実装を分離

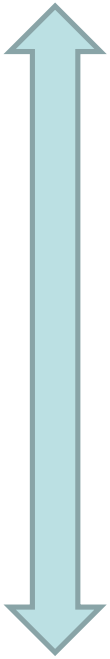
カプセル
(capsule)

- モジュール強度（凝集度）(module strength/cohesion)
  同じモジュール内に存在する構成要素の関連の程度
- モジュール結合度(module coupling)
  異なるモジュール間に存在する構成要素の関連の程度

モジュール強度(凝集度)

- 暗号的強度、偶発的強度 (coincidental cohesion)
- 論理的強度(logical cohesion)
- 時間的強度(temporal cohesion)
- 手順的強度(procedural cohesion)
- 連絡的強度(communicational cohesion)
- 情報的強度(informational cohesion)
- 機能的強度(functional cohesion)

凝集度低



凝集度高

※モジュール強度の強さの順番は解釈によって変わることもある

モジュール強度(凝集度)

(1)暗号的強度、偶発的強度(coincidental cohesion)

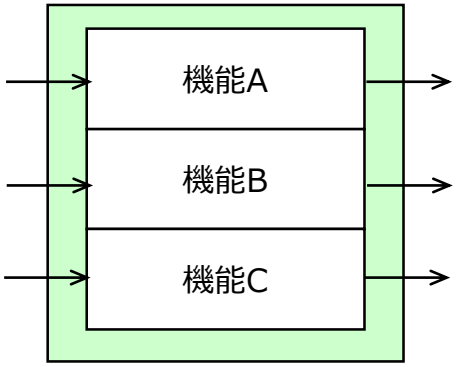
構造的関係、意味的関係がほとんど(まったく)ない処理を集めたモジュール

(2)論理的強度(logical cohesion)

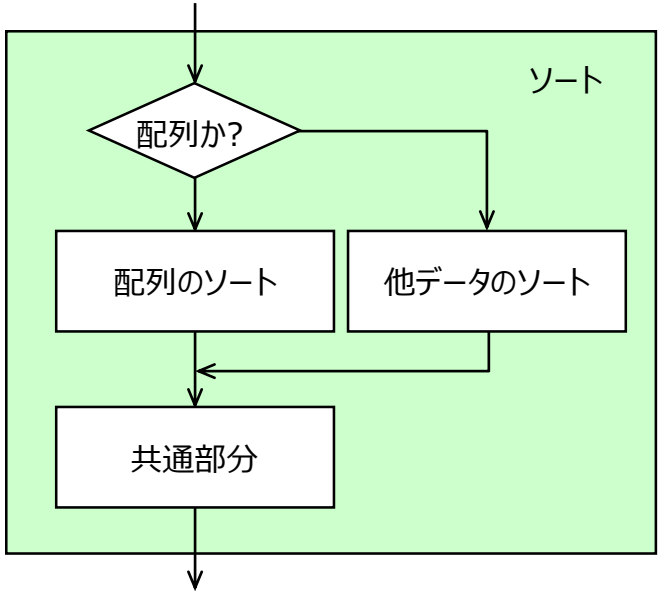
見かけ上は同一の機能を持つが、実際には多様な機能を集めたモジュール
(e.g., 並べ替えモジュール、エラー処理モジュール)
数学計算ライブラリ等は論理的強度だが、一般に悪い設計ではない

(3)時間的強度(temporal cohesion)

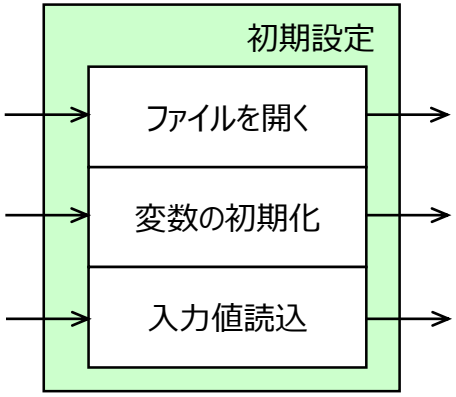
実行されるタイミングが近い機能を集めたモジュール



暗号的



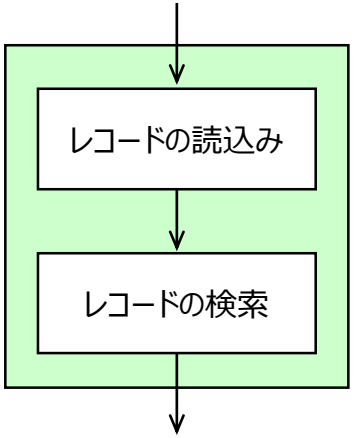
論理的



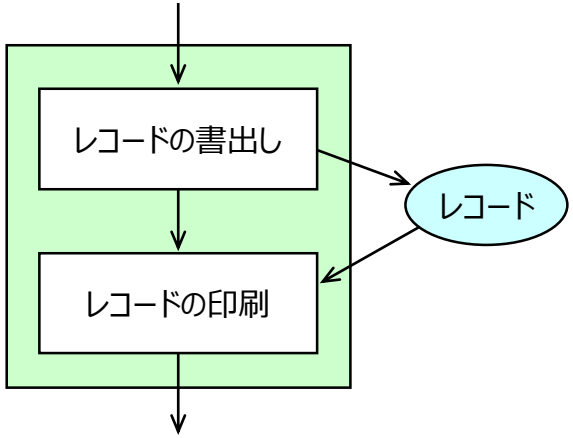
時間的

モジュール強度(凝集度)

(4)手順的強度、手続き的強度(procedural cohesion)
　逐次的に実行される関連のある機能を集めたモジュール
(5)連絡的強度、通信的強度(communicational cohesion)
　手順的強度で、
　同じデータを入力あるいは出力する機能を集めたモジュール



手順的



連絡的

モジュール強度(凝集度)

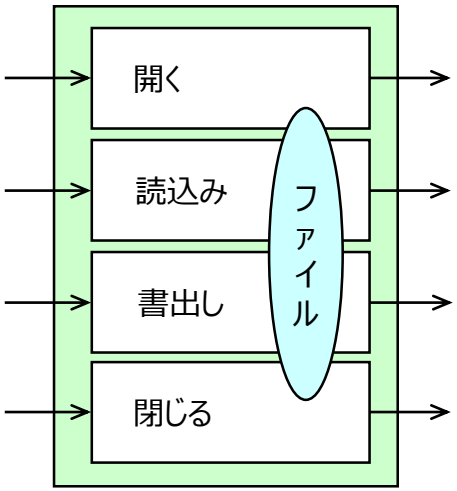
(6)情報的強度(informational cohesion)

同じデータにアクセスする複数の機能を集めたモジュール
(オブジェクト指向のクラスが該当)

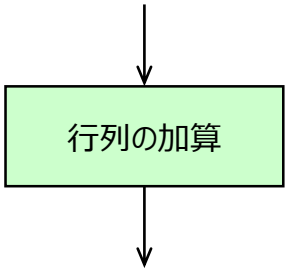
(7)機能的強度(functional cohesion)

単一の機能を実行するモジュール

(6)が最も良いとする考え方もある



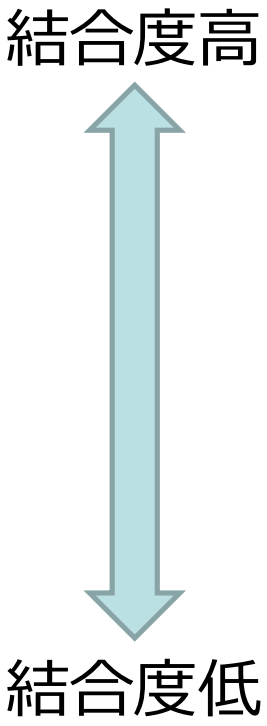
情報的



機能的

モジュール結合度

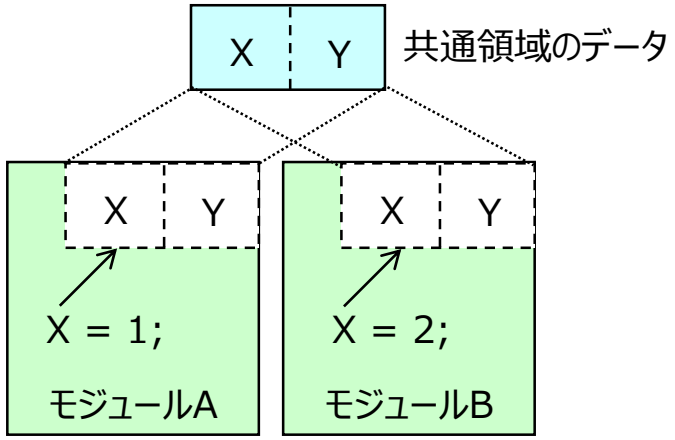
- 内容結合(content coupling)
- 共通結合(common coupling)
- 外部結合(external coupling)
- 制御結合(control coupling)
- スタンプ結合(stamp coupling)
- データ結合(data coupling)



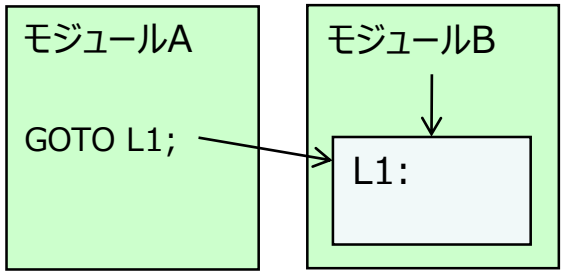
※モジュール結合度の強さの順番は解釈によって変わることもある

モジュール結合度

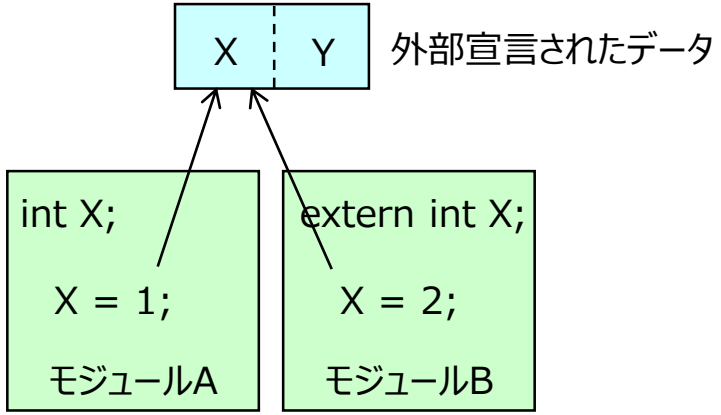
(1)内容結合(content coupling)
　一方のモジュールが他方のモジュールの内容を直接参照したり更新したりする
(2)共通結合(common coupling)
　モジュール同士が共通データ領域にあるデータを参照する
(3)外部結合(external coupling)
　モジュール同士が外部宣言されたデータを共有する



共通結合



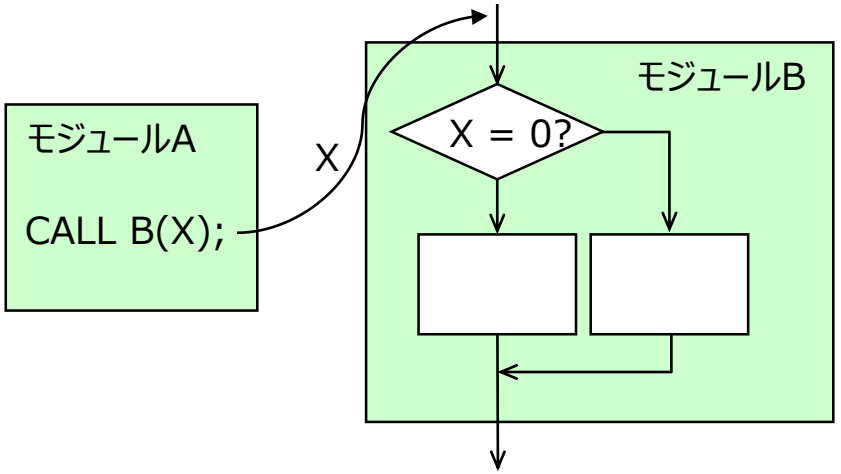
内容結合



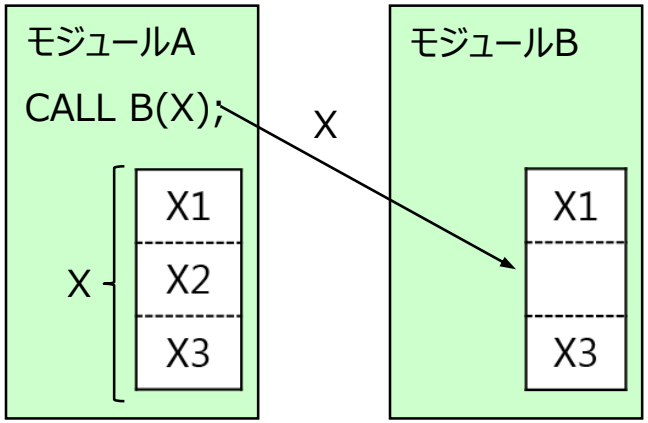
外部結合

モジュール結合度

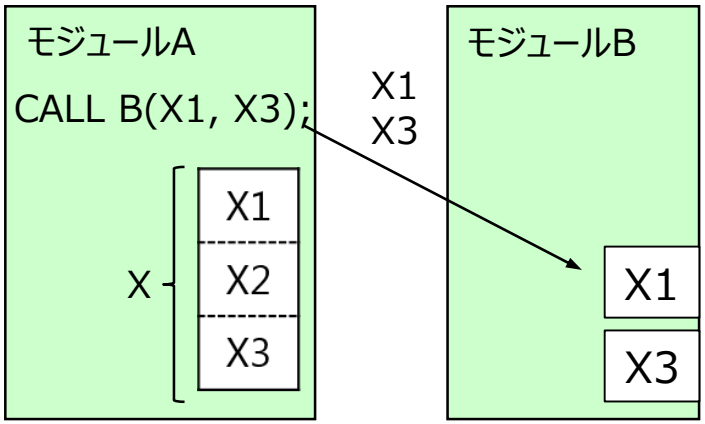
(4)制御結合(control coupling)
　引数に基づいて条件判断を行う
(5)スタンプ結合(stamp coupling)
　共通データ領域にないデータの
　構造体やオブジェクトを
　受け渡す (不要なデータも含まれる)
(6)データ結合(data coupling)
　必要なデータだけを引数として受け渡す



制御結合



スタンプ結合



データ結合

確認問題

括弧内に選択肢があるものは正しいものを選択

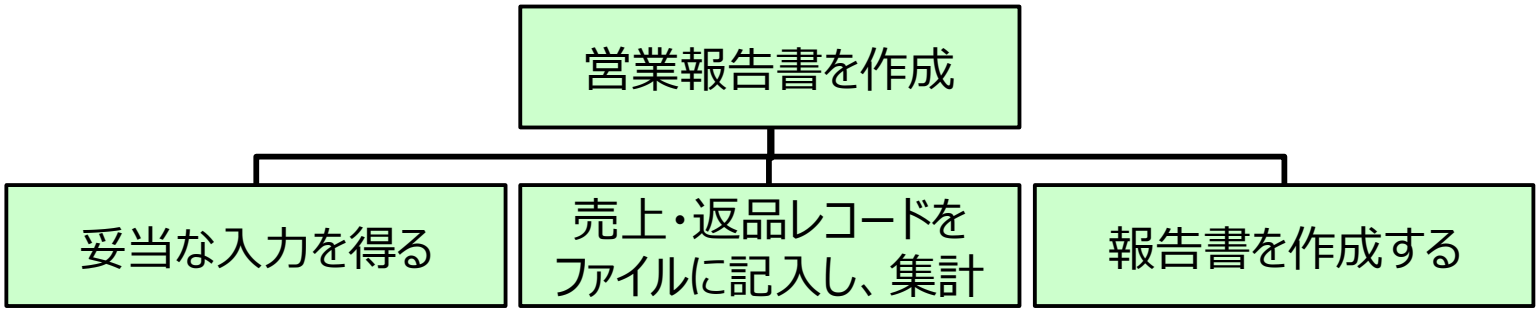
- (　1　)は、一連の機能をひとまとめにしたプログラムの構成単位である。
- 一般的に、(　1　)は(2 複数・単数)の文で構成され、決められた(　3　)を通してのみ呼出し可能である。
- モジュール分割の利点としては、コードの(4 具体・抽象)化や、開発効率、再利用性、変更容易性の向上が挙げられる。
- 一般的に、モジュール結合度は(5 高い・低い)方が良いとされる。
- 一般的に、モジュール強度は(6 高い・低い)方が良いとされる。

講義内容

- モジュール設計
  - 概説
  - モジュール分割の評価基準
  - モジュール分割技法
  - オブジェクト指向におけるモジュール設計(参考)

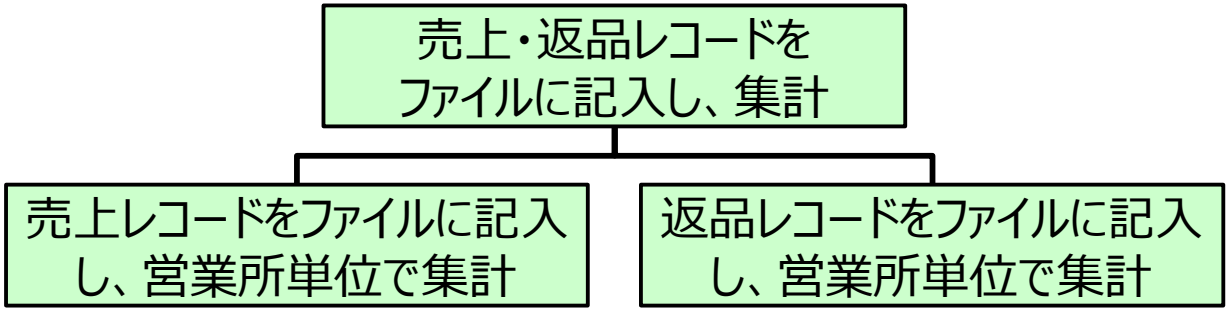
モジュール分割技法

- STS分割
  - 源泉(source)：データ入力部
  - 変換(transform)：データ変換部
  - 吸収(sink)：データ出力部
  - S,T,Sを担当するモジュールをそれぞれ配置
    - データフロー上のデータ変換を追跡し、入力データ・出力データとはみなせなくなった所で分割
  - それぞれを制御するモジュールを配置



モジュール分割技法

- TR分割(トランザクション分割)
  - トランザクション処理：入力データに関する一連の処理
    - データベースの一貫性を維持
    - 途中で終わらせたり、一部だけ実行することはできない
  - 機能をトランザクションごとに分割
    - 入力データの種類が複数あり、それぞれ異なる処理を行う場合に用いられる



共通機能分割

- 他の分割方法で機能分割を行う過程で、
  共通する機能を別個のモジュールとして抽出
- 共通のデータに関する機能を1個のモジュール
  として抽出
- ユーティリティとして使う機能群を抽出する
  際に有効

確認問題

- 以下の説明に合うモジュール分割技法を答えよ。
  - 複数のモジュールに共通する機能を別のモジュールとして抽出する。
  - 機能を、途中で終わらせたり、一部だけ実行することはできない単位に切り分け、モジュールとする。
  - データの入力、変換、出力それぞれを担当するモジュールを作成する。

講義内容

- モジュール設計
  - 概説
  - モジュール分割の評価基準
  - モジュール分割技法
  - オブジェクト指向におけるモジュール設計(参考)

オブジェクト指向設計における モジュール強度・結合度

- クラス
  - 情報的強度：
    同一データにアクセスする機能をまとめている
  - 理解容易性、変更容易性のため、
    複数のデータを1クラスにまとめる場合もある
    → 情報的強度よりも低下
  - 2つのクラスがデータ結合であっても、
    クラス間のやり取りが多いのは望ましくない
- パッケージ
  - 複数のクラスをまとめている
    → これもモジュールの一種

オブジェクト指向では、
従来のモジュール強度・結合度とは違った指針が必要

クラスの設計原則

- 単一責任の原則 (SRP: single responsibility principle)
  - クラスを変更する理由は1つでなければならない
- オープン・クローズドの原則 (OCP: open-closed principle)
  - 拡張に対してオープンで、修正に対してクローズで
    なければならない
- リスコフの置換原則 (LSP: Liskov substitution principle)
  - サブクラスはそのスーパークラスと置換可能でなければならない
- 依存関係逆転の原則 (DIP: dependency inversion principle)
  - 上位のモジュールは下位のモジュールに依存してはいけない
  - 抽象は実装の詳細に依存してはいけない
- インタフェース分離の原則 (ISP: interface segregation principle)
  - 強い関連性を持つインタフェースのみをまとめて
    グループ化しなければならない

パッケージの設計原則

- 再利用・リリース等価の原則 (reuse-release equivalence)
  - パッケージはリリースの単位で再利用されなければならない
- 閉鎖性共通の原則 (common closure)
  - 1つの変更理由は単一パッケージに閉じ込められなければならない
- 全再利用の原則 (common reuse)
  - パッケージ内の全クラスが再利用されなければならない
- 非環式依存の原則 (acyclic dependencies)
  - パッケージの依存関係は無閉路有向グラフでなければならない
- 安定依存の原則 (stable dependencies)
  - より安定しているパッケージに依存しなければならない
- 安定度・抽象度等価の原則 (stable-abstraction)
  - 抽象的なパッケージほど安定していなければならない

参考文献

- 「ソフトウェア工学」
  高橋直久、丸山勝久 著、森北出版、2010
- 「効果的プログラム開発技法」
  國友義久 著、近代科学社、1979
- 「ソフトウェア工学」
  中谷多哉子、中島震 著、放送大学教育振興会、2019
- R.C. Martin et al., Principles of object oriented design
  http://wiki.c2.com/?PrinciplesOfObjectOrientedDesign