# オブジェクト指向とモジュール性

TU 坂田誠也

# 目次

# 概要

- 話すこと
  - 変更を抑えつつ機能拡張できないか考えよう。
  - 実装の詳細はモジュール自体をドキュメントとしよう。
  - モジュールの責務をできる限り分けるよう意識しよう。
  - 一貫性のある表記を心がけよう
- 話さないこと
  - 他の各論の詳細（抽象データ型、契約による設計、表明・例外）
  - オブジェクト指向プログラミングの手法

# テーマの動機

自分が一からソフトウェア設計する際に、変更に強くて拡張性の高い設計ができるようになっておきたい。💪

# テーマの動機

ソフトウェア設計で最も汎用的な書籍と言えば、「Clearn Architecture 達人に学ぶソフトウェアの構造と設計」かもしれない。


[Clean Architecture 達人に学ぶソフトウェアの構造と設計
Robert C. Martin(著), 角征典, 髙木正弘(訳)]

# テーマの動機

Clean Architecture 達人に学ぶソフトウェアの構造と設計「序文」より

**アーキテクチャのルールはどれも同じである!**

**書いているコードが変わらないのだから、どんな種類のシステムでもソフトウェアアーキテクチャのルールは同じ。ソフトウェアアーキテクチャのルールとは、プログラムの構成要素をどのように組み立てるかのルールである。構成要素は普遍的で変わらないのだから、それらを組み立てるルールもまた、普遍的で変わらないのである。**

# テーマの動機

とは言え、Clean Architecture や関数型プログラミングもオブジェクト指向があってのものなので、まずオブジェクト指向に対する正しい理解が重要。

# テーマの動機

オブジェクト指向とは何かと聞かれて答えられるか？🤔

オブジェクト指向プログラミングだとカプセル化とか継承とかインターフェースなどのことだけど、それがどうしてオブジェクト指向的なのか？

本当に「分かってる」か？？？

# テーマの動機

ということで入門（？）しました。
オブジェクト指向入門 第 2 版 原則・コンセプト

# テーマの動機

全体で**960**ページ（！）

しかも下巻相当がありこれで全体の半分...

なので今回は各論として「モジュール性」について話します。

# 自己紹介

坂田誠也(27)

興味: 設計、セキュリティ、プログラミング言語
趣味: ゲーム、アニメ、麻雀
その他: サッカー 5 年、ピアノ 7 年、ドラム 3 年、韓国語 6 年、フランス語 3 年

# ソフトウェアの品質の定義

Q. モジュール性とは？

A. 拡張性と再利用性のこと

Q. では拡張性と再利用性とは？

答えの前にソフトウェアの品質の話から

# ソフトウェアの品質の定義

ソフトウェアの品質とはどう定義できるか？

- バグが発生しないこと？
- 使いやすいこと？
- スピード？

# ソフトウェアの品質の定義

大きく分けて2つ

- 外的品質要因
- 内的品質要因

# ソフトウェアの品質の定義

- **外的品質要因** ←
- 内的品質要因

# ソフトウェアの品質の定義

## 外的品質要因

ユーザーに認識される品質。

例）

- 正確さ
- 頑丈さ
- 拡張性
- 再利用性

# ソフトウェアの品質の定義

- 外的品質要因
- **内的品質要因** ←

# ソフトウェアの品質の定義

## 内的品質要因

エンジニアのみが認識する品質。

例）

- 可読性
- テスト
- メモリ管理

# ソフトウェアの品質の定義

最終的に重要なのは外的品質要因、すなわちユーザーからの評価に直結する部分だが、

**外的品質要因を達成する鍵は内的品質要因にある。**

# ソフトウェアの品質の定義

今回のテーマとなる外的品質要因は、特に重要な

**拡張性**
**再利用性**

です

# ソフトウェアの品質の定義

## 拡張性

伝統的なソフトウェア工学手法では、最初の要件を固定して、残りで設計と実装に費やすのを前提としていた。

現在のソフトウェア開発では「変わる」ことを前提とすることが重要になってきた。

- 要件
- アルゴリズム
- データ形式 etc...

# ソフトウェアの品質の定義

## 拡張性

## 拡張性を向上させる原則

- 設計の単純さ
  - 単純なアーキテクチャは常に複雑なアーキテクチャよりも変更に適応しやすい。
- 非集中化
  - モジュールの独立性が高いほど変更の影響が最小限に抑えやすい。

# ソフトウェアの品質の定義

## 再利用性

共通パターンを把握することで

- 再利用可能なモジュールを他の異なる開発に応用できる。
- 実質的に実装コストが減るので、同じコストで他の品質要因に割ける。

# オブジェクト指向でのモジュール性とは

オブジェクト指向において、モジュール性は

「5 つの基準」から「5 つの規則」が導かれ、最終的に「5 つの原則」が導かれる。

# モジュール性の 5 つの基準

- 分解しやすさ
- 組み合わせしやすさ
- 分かりやすさ
- 連続性
  - ある変更が発生したときに、影響が最小限に抑えられること。
- 保護性
  - エラーや例外が発生したときに、影響が最小限に抑えられること。

# モジュール性の 5 つの規則

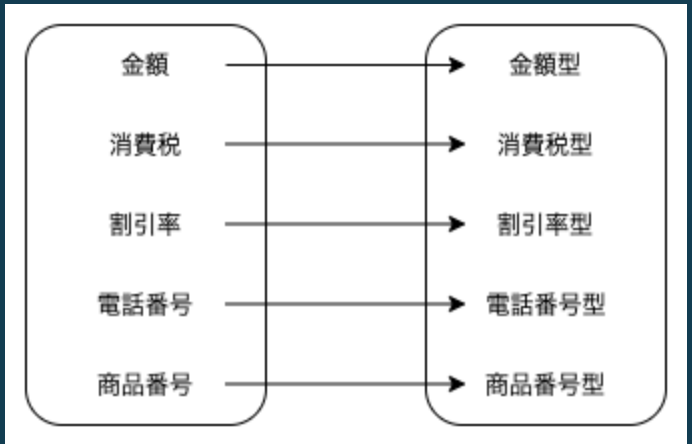
- 直接的な写像
- 少ないインターフェース
- 小さいインターフェース
- 明示的なインターフェース
- 情報隠蔽

# モジュール性の 5 つの規則

## 直接的な写像

***そのソフトウェアで解決する領域の構造とソフトウェアのモデルの構造の互換性を維持する***

ソフトウェアシステムで構築されるモデルは、そのソフトウェアで解決する領域の構造との互換性を維持する。



# モジュール性の 5 つの規則

## 直接的な写像

以下の基準から導かれる。

- 分解しやすさ
- 連続性

# モジュール性の 5 つの規則

## 少ないインターフェース

***外部接続を行うモジュールはできる限り少なくする。***

通信を行うモジュールが多いと変更とエラーの影響を受けやすくなる。

以下の基準から導かれる。

- 連続性
- 保護性

# モジュール性の 5 つの規則

## 小さいインターフェース

***モジュール間で通信するデータサイズはできるだけ少なくする。***

こちらも通信をするデータのサイズが小さいと変更とエラーの影響を受けやすくなる。

以下の基準から導かれる。

- 連続性
- 保護性

# モジュール性の 5 つの規則

## 明示的なインターフェース

***モジュール間で外部接続を行う場合、外部インターフェースがお互いから見て明確になっていること。***

モジュールの外部との接続部分（インターフェース）が明示的になると、変更が明確になり、分解・組み立てが容易になる。

# モジュール性の 5 つの規則

## 明示的なインターフェース

以下の基準から導かれる。

- 分解しやすさ
- 組み合わせしやすさ
  - モジュールの外部との接続部分が明確になると分解・組み立てが容易になる。
- 連続性
  - 発生する変更でどこが変更を受けるか明確になる。

# モジュール性の 5 つの規則

## 情報隠蔽

***モジュールは機能の仕様を公開し、実装は隠蔽することで変更の影響を抑えること。***

publicな属性が少ないほど、そのモジュールで変更があった場合に非公開の属性で変更が発生している可能性が大きくなる。

# モジュール性の 5 つの規則

## 情報隠蔽

例）カーナビの経路検索機能

現在地や目的地の住所はユーザーが使うのでpublicな属性だけど、探索アルゴリズムや渋滞情報などは内部の手続きなのでprivateな属性になる。
ユーザーが外部インターフェースのお作法を守っていれば、アルゴリズムや渋滞情報の取得方法が変わっても、privateな属性なのでユーザーに影響を与えることはない。

# モジュール性の 5 つの原則

- 言語としてのモジュール単位の原則
- 自己文書化の原則
- 統一形式アクセスの原則
- 開放・閉鎖の原則
- 単一責任原則

# モジュール性の 5 つの原則

## 言語としてのモジュール単位の原則

***モジュールは使用される言語の構文単位に対応していなければならない。***

ここでの言語は、プログラミング言語に限らず仕様・設計などでの表現のことです。

仕様から実装に至るまでに、モジュールが明確で一貫性を持って分割されていることが大切です。

# モジュール性の 5 つの原則

## 言語としてのモジュール単位の原則

例）

用語の定義

命名規則

# モジュール性の 5 つの原則

## 言語としてのモジュール単位の原則

- 連続性
  - 仕様・設計・実装それぞれのレイヤーで一貫性を保つ。
- 直接的な写像
  - 解決領域の構造とソフトウェアのモデルの構造の互換性を保つ。
- 分解しやすさ
- 組み合わせやすさ
- 保護性

# モジュール性の 5 つの原則

## 自己文書化の原則

***モジュールについてのすべての情報をそのモジュールの一部として作ること。***

変更がたくさん発生すると、ドキュメントとソフトウェアが食い違うことが発生する危険がある。

モジュール自体がドキュメントのすべてになることはないが、実装の詳細はドキュメントには不向きなのでモジュールに含めるべき。

- 情報隠蔽

# モジュール性の 5 つの原則

## 統一形式アクセスの原則

***あるモジュールによって提供されるサービスはすべて統一された表記によって利用できなければならない***

ある操作をするときに、それがプロパティであるかメソッドであるかを利用者側が気にすることなく同じ表現で利用できること。

# モジュール性の 5 つの原則

## 統一形式アクセスの原則

Rubyの場合

```ruby
class Point
  def longitude
    @longitude
  end
  def longitude=(lon)
    @longitude = lon
  end
end
p = Point.new
p.longitude = 135
puts p.longitude # 135
```

# モジュール性の 5 つの原則

## 開放・閉鎖の原則

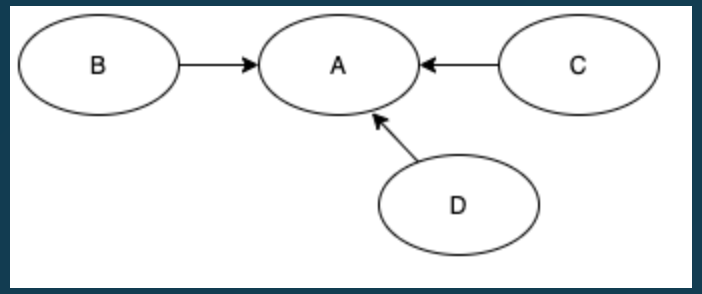
***モジュールは開いていると同時に閉じているべきである。***

- モジュールが拡張可能な状態は、そのモジュールは開放されていると言える。
- モジュールが変更の影響を受けない状態にある場合、そのモジュールは閉鎖されていると言える。

# モジュール性の 5 つの原則

## 開放・閉鎖の原則

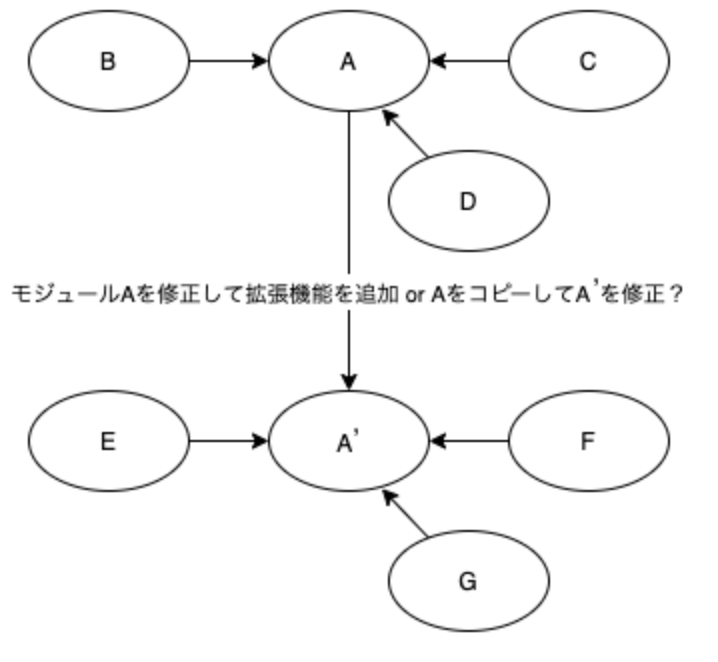
例）
あるモジュールAが複数のモジュールから使用されているとする。



# モジュール性の 5 つの原則

## 開放・閉鎖の原則

Q.ここに新たなモジュールE、F、Gが追加され、拡張されたモジュールA（モジュールA’と呼ぶ）が要求されたとする。どうすればいいか



# モジュール性の 5 つの原則

## 開放・閉鎖の原則

オブジェクト指向ではない手法の場合

1. モジュールAを修正して、新しい要求に合う拡張されたモジュールAとしてモジュールA’を提供する。
2. モジュールAをそのままにコピーを作って、新しいモジュールに必要な変更を追加する。（モジュールAとモジュールA’はもはや無関係）

# モジュール性の 5 つの原則

## 開放・閉鎖の原則

1の場合: Aの変更がモジュールBやモジュールCへ影響、さらにモジュールBやモジュールCを使っているモジュールがあれば連鎖的に変更が波及する。

2の場合: 最悪

# モジュール性の 5 つの原則

## 開放・閉鎖の原則

A. 継承を行う

モジュールAを継承してモジュールA'を定義すれば、モジュールAを変更する必要がなくなる。

詳しく話すと時間があまりにも足りないので...

**注）継承 = 開放・閉鎖の原則ではなく、あくまで一例です。**

# モジュール性の 5 つの原則

## 単一責任原則

***一つのモジュールに対して一つの機能しか持たない。***

単一責任原則は開放・閉鎖の原則と情報隠蔽の原則の両方から導かれる。
クラスを変更する理由は複数存在してはいけないとも言える。

# モジュール性の 5 つの原則

## 単一責任原則

例）図書館システムの出版物を表すデータ構造

ISBN、著者、出版社、etc...

例えばこれが同じモジュールに全部ある場合、変更が発生した場合にこのモジュールは著者や出版社もまとめて影響が発生しないか確認する必要がある。

これを避けるためには、ISBN、著者、出版社などを分離することで変更の影響を抑えることができる。

# まとめ

- 変更を抑えつつ機能拡張できないか考えよう。
- 実装の詳細はモジュール自体をドキュメントとしよう。
- モジュールの責務をできる限り分けるよう意識しよう。
- 一貫性のある表記を心がけよう