

クラス設計概説

なぜオブジェクト指向(クラス設計)を学ぶのか

- クラス設計をする機会は実は少ない
 - 仕事ではフレームワークを使ってコーディングをする場合がほとんど
⇒オブジェクト指向で設計された結果を利用しているだけ
- クラス設計を理解していないと...
 - フレームワークの設計の意図が良く分からない
 - その結果、他人がソースコードを読んでも内容が分からないことに
 - バグが多い、修正に手間がかかる
 - 機能追加するときに、既存コードへの影響が判断できない

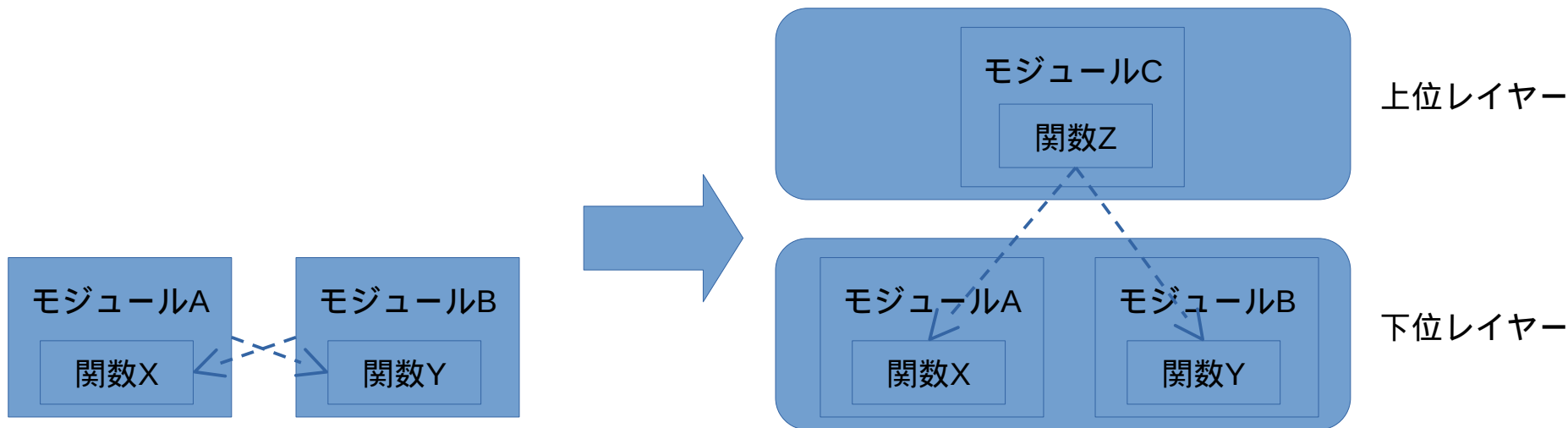
プログラミング設計の基本（言語共通）

- モジュール分割

- 機能を分割する⇒モジュール（Javaのクラスに相当）に分ける
- モジュールの独立性を高める ⇒ 他のモジュールが変更されてもプログラム修正不要となるようにする
 - それぞれのモジュールの機能を単純化する
 - 公開機能と非公開機能の区別をする
 - 余計な情報を他のモジュールに公開しない
 - モジュール間の依存関係（呼び出し関係）を最小限にする
 - 呼び出し先のモジュールになるほど機能を単純化する
 - 相互呼び出しの排除 ⇒ モジュール間の呼び出しは一方通行
 - モジュールのレイヤー化 ⇒ 類似性の強いモジュールをグループ化（レイヤー）して、レイヤー間で呼び出し関係が一方向になるようにする

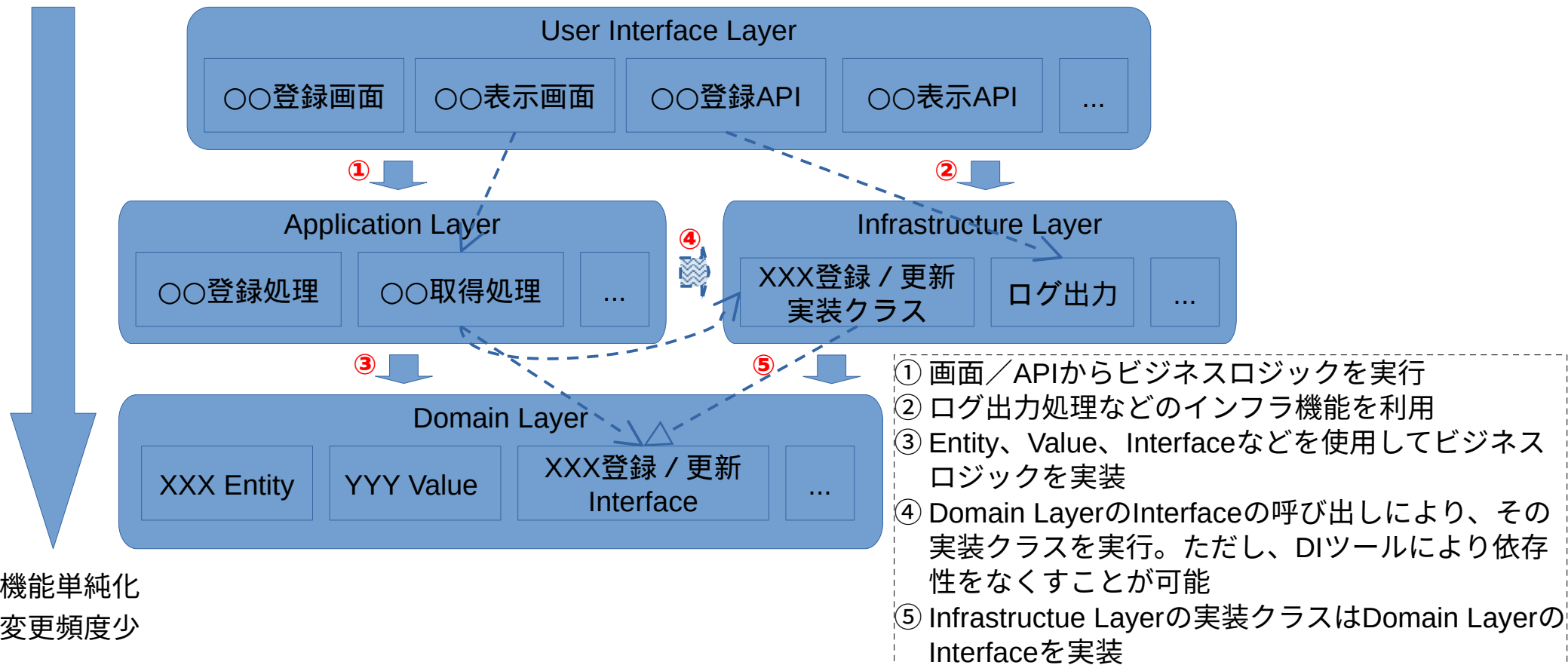
相互呼び出しの排除

- モジュールAとモジュールBが相互に呼び出す関係の場合
 - 例えばモジュールBからモジュールAのX関数を呼び出した後に、モジュールAからモジュールBのY関数を呼び出している場合
 - A、Bの上位モジュールとしてモジュールCを作成
 - モジュールCからモジュールAのX関数を呼び出した後にモジュールCからモジュールBのY関数を呼び出すようにする



レイヤーの例（Webアプリケーション）

- Layer間の呼び出し方向を一方向にすることで、モジュール全体の依存関係が整理しやすくなる



公開機能と非公開機能

- 何故2つに分けるのか？
 - 公開機能は一度公開するとAPIは変更できない
 - 公開する機能やデータは最小限にしてAPI変更のリスクを低減する
 - 非公開機能は公開機能を実現するためのロジックを実装
 - 要件の追加やリファクタリングなどで実装が変わる可能性が大きい
 - 公開してしまうとロジック変更により呼び出し元もコード修正が発生
⇒「強い依存関係」が発生するので良くない

オブジェクト指向の考え方

- モジュールの関係を物(Object)に例えて整理する
 - 抽象的なままでは整理しにくいので具体的な形に変えて整理する
 - 最初は利用者、ATM、通帳、キャッシュカード、現金、口座などの具体的なObjectから考える
 - 具体的な形が整理出来たら、一般化（抽象化）して設計を進める
- オブジェクトのメッセージ
 - オブジェクトの機能呼び出すことを「メッセージ」という
 - 例：「利用者がATMから現金を引き出す」⇒「利用者」オブジェクトが「ATM」オブジェクトに「現金を引き出す」メッセージを送る
 - メッセージは、クラスのメソッドで実現（実装）される

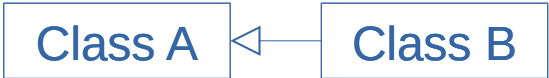
オブジェクトとクラスの違い

- オブジェクト
 - 実際に存在する「もの」
 - クラスの実体（インスタンス）に相当する
- クラス
 - オブジェクトを作る基になる設計情報
 - 1つのクラスから複数のインスタンスが作成される
 - 人が10人は、「人」クラスのインスタンスが10個ということ


アクセス制御

- アクセス修飾子(public、private、etc)
 - アクセス修飾子によって他のクラスからアクセス可能な範囲を制限する
 - publicメソッドはすべてのクラスから呼び出し可能 ⇒ 公開機能
 - privateメソッドは、他のクラスからの呼び出し不可 ⇒ 非公開機能


クラス間の関係とクラス図 (1)

- is-a関係 

```
classDiagram
    ClassA <|-- ClassB
```

 - 「Class BはClass Aでもある」という意味
 - つまり「Class BはClass Aを継承している」ということ
- has-a関係 

```
classDiagram
    ClassA *-- ClassB
```

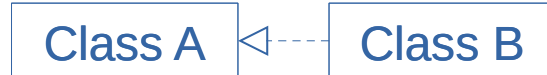
 - 「Class AはClass Bを持っている」という意味
 - つまり「Class BはClass Aの部分クラスである」ということ
 - もっと言うと、Class BのインスタンスがClass Aの属性に存在する
- 依存関係 

```
classDiagram
    ClassA ..> ClassB
```

 - 「Class BはClass Aに依存している」という意味
 - つまり「Class BはClass Aのメソッドを呼び出している」ということ

クラス間の関係とクラス図 (2)

- インターフェースと実装クラス



- 「Class BはInterface Aを実装している」という関係
- Interface AはAPI（メソッドの引数と戻り値）を定義するだけで、基本的に実装（メソッドのコード）がない
- Class Bは、Interface Aで定義されたAPIを実装する（メソッドのコードを書く）
- Javaで多態性（ポリモフィズム）を実現する方法
 - 例えばJDBCの場合は、APIをインターフェースで定義して、DB製品毎に実装クラスを提供している
 - 利用者は、JDBCのインターフェースを使用することにより、実装（DB製品）の違いを気にせずにコーディングができる ⇒ 使用するDB製品を変えても利用者はソースコードの修正をしなくて良い