オブジェクト指向概説

2025.01.19 金田篤実

目次

- オブジェクト指向とは
- オブジェクト指向を実現するための仕組み
- オブジェクト指向の5大原則
- デザインパターンとは
- デザインパターンを学ぶことのメリット

オブジェクト指向とは

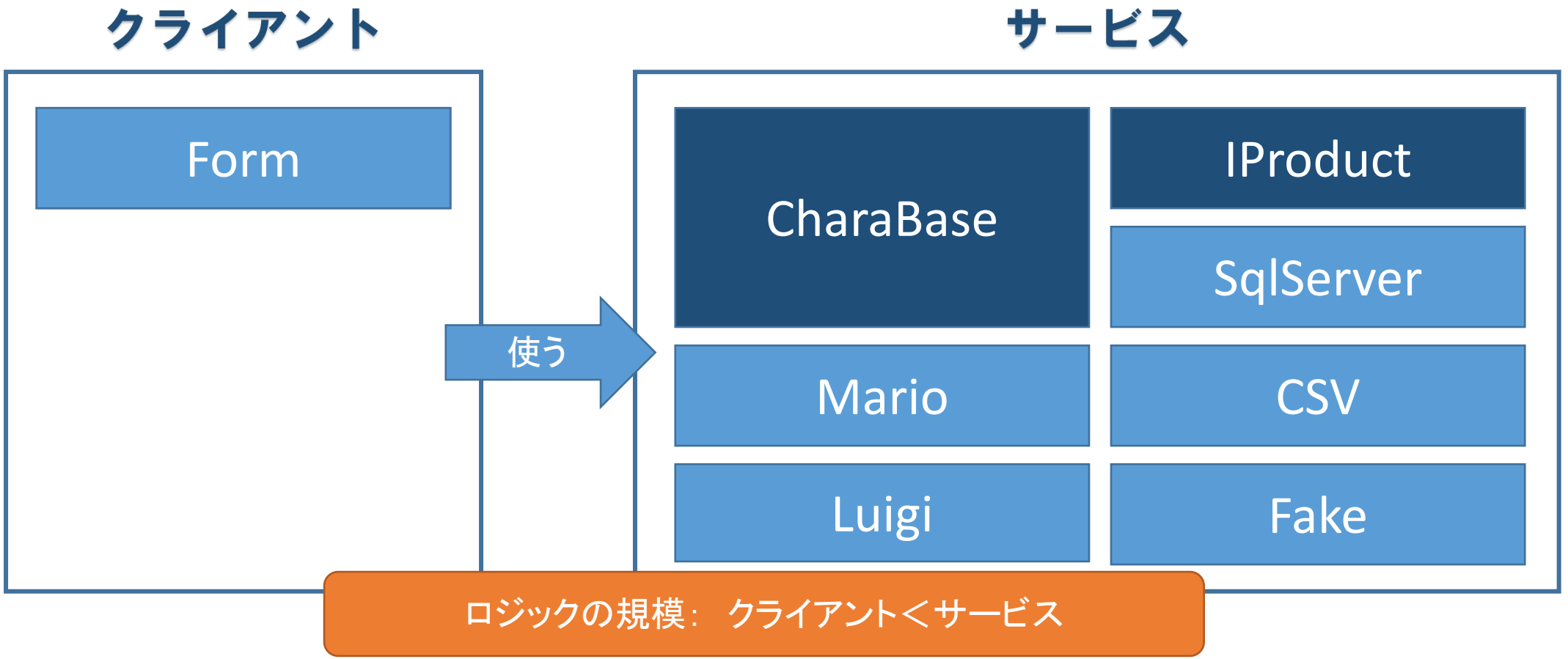
正しい定義はよくわかりませんが、自分が思うわかりやすい定義を示します。

オブジェクト指向(プログラミング)：

プログラムを「クライアント」と複数のオブジェクトで構成された「サービス」に分け、大筋の処理はクライアントに、詳細な処理はサービスの各オブジェクトに記述する手法。クライアントコードは極力、複雑な条件分岐などの処理を極力少なくした抽象的なコードにし、各オブジェクトに具体的な処理を任せることにより、仕様変更時のコード修正範囲を少数のオブジェクトに限定することができる。

オブジェクト指向とは

イメージ図



オブジェクト指向を実現するための仕組み

オブジェクト指向(プログラミング)を実現するためには下記の仕組みが必要になります。

・データのカプセル化：

オブジェクト内部で使用するデータを外部からアクセスできないようにする。

→オブジェクトの独立性が高まりコード変更範囲が局所化される。

・振る舞いのカプセル化：

オブジェクトの操作を抽象化し外部から他オブジェクトと同様に扱えるようにする。

(同じメソッドをコールしてもオブジェクト毎に振る舞いが異なる＝ポリモーフィズム)

→オブジェクトを使用する側のコードが抽象化されコード変更が少なくなる。

オブジェクト指向の5大原則

・単一責務の原則
・オープンクローズドの原則
・リスコフの置換原則
・依存関係逆転の原則
・インターフェイス分離の原則

オブジェクト指向の5大原則

単一責務の原則：

クラスの責務は一つであり、そのクラスの変更理由はその責務の変更のみで
なければならない。(責務＝変更理由)

解説：

「責務」の定義が曖昧なので、なるべくクラスを細分化した方が変更箇所が特定しやすい&変更箇所が少ないプログラムになる、程度の理解で良いと思います。
小さなクラスとしてはValueObjectが良い例である。

オブジェクト指向の5大原則

オープンクローズドの原則：

既存のコードは変更せずに、プログラムの機能拡張ができるように設計するのがベター。

(拡張に対してオープン、変更に対してクローズドという意味)

解説：

インターフェイスや抽象クラスを使用してオブジェクトの振る舞いをカプセル化し、なるべくクライアントコードを抽象的に設計することで実現します。

これにより、既存のクライアントコードやオブジェクトは変更せずに、新しいオブジェクトを追加するだけで機能拡張することが可能となります。

オブジェクト指向の5大原則

リスコフの置換原則：

サブクラスは基底クラスと置換可能でなければならない。

(サブクラスでは基底クラスにないプロパティやメソッドを新たに定義しない。

基底クラスに全ての定義があり、サブクラスはその一部を上書きするだけ。)

オブジェクト指向の5大原則

(リスコフの置換原則の続き)

解説：

同じ基底クラスから継承されたMarioクラスとLuigiクラスがあり、Luigiクラスにしかない定義(プロパティやメソッド)があった場合、クライアント側では、それらの定義にアクセスするためにはLuigiクラスかどうかを識別する必要性＝クライアント側に知識が必要になります。(この場合はLuigiクラスを使う、この場合はMarioクラスを使う…など)

これは、クライアントの抽象度を下げ仕様変更による影響を受けやすくなってしまうためNGです。

オブジェクト指向の5大原則

(リスコフの置換原則の続き)

リスコフの置換原則のポイント：

・クライアントから見て、基底クラスとサブクラスを同一視できない継承はNGである。

・継承は、基底クラスの機能が欲しいからするのではなく、基底クラスの動作を置き換えるためにするもの。

・クライアントは基底クラスに対してコーディングされるべきであり、クライアントはどのサブクラスのプロパティやメソッドにアクセスしているかを、知るべきでないし知る必要もない。それを知ってしまうとクライアントもオブジェクト側のコード変更の影響を受けることになってしまう。

オブジェクト指向の5大原則

依存関係逆転の原則：

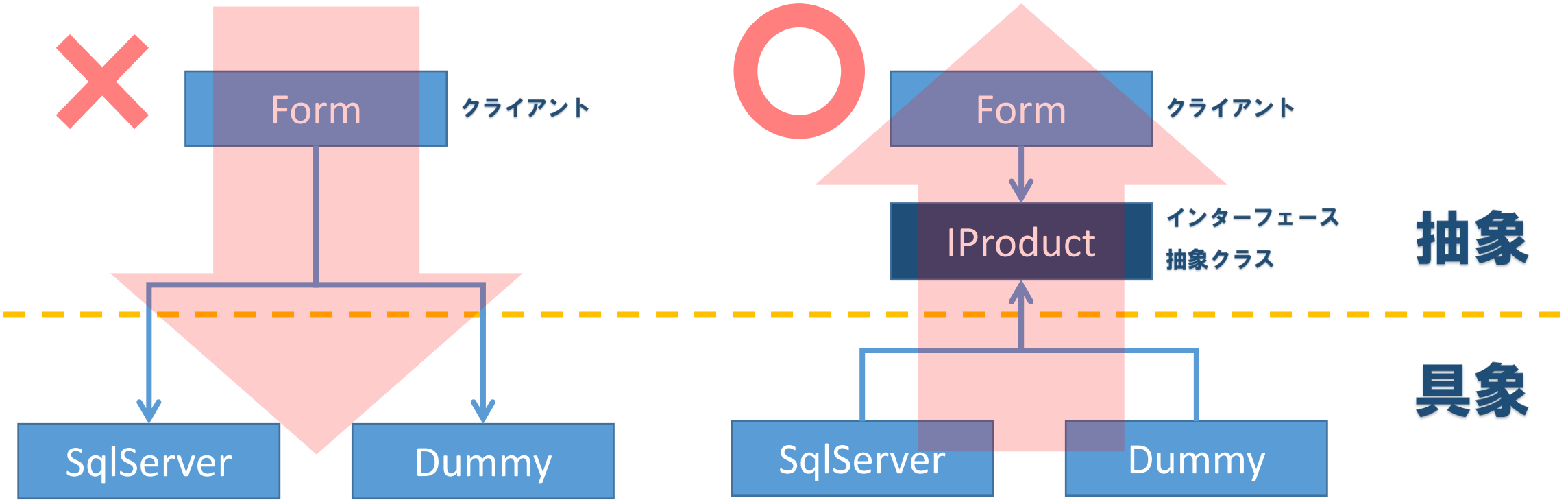
上位レベルのモジュールは下位レベルのモジュールに依存すべきではない。
両方とも抽象に依存すべきである。

解説：

上位/下位レベルのモジュールは、インターフェースや抽象クラスといった抽象に依存するように実装すべきであるということ。これを疎かにすると、上位レベルのモジュール＝主にクライアントコードはいつまでたっても具象を担当する下位モジュールに依存する形となる。結果として上位モジュールは抽象的にならず仕様変更の影響範囲にさらされる確率が高くなる。

オブジェクト指向の5大原則

(依存関係逆転の原則の続き)



オブジェクト指向においては、抽象が具象に依存するのではなく、具象が抽象に依存しなければならない。

オブジェクト指向の5大原則

インターフェイス分離の原則：

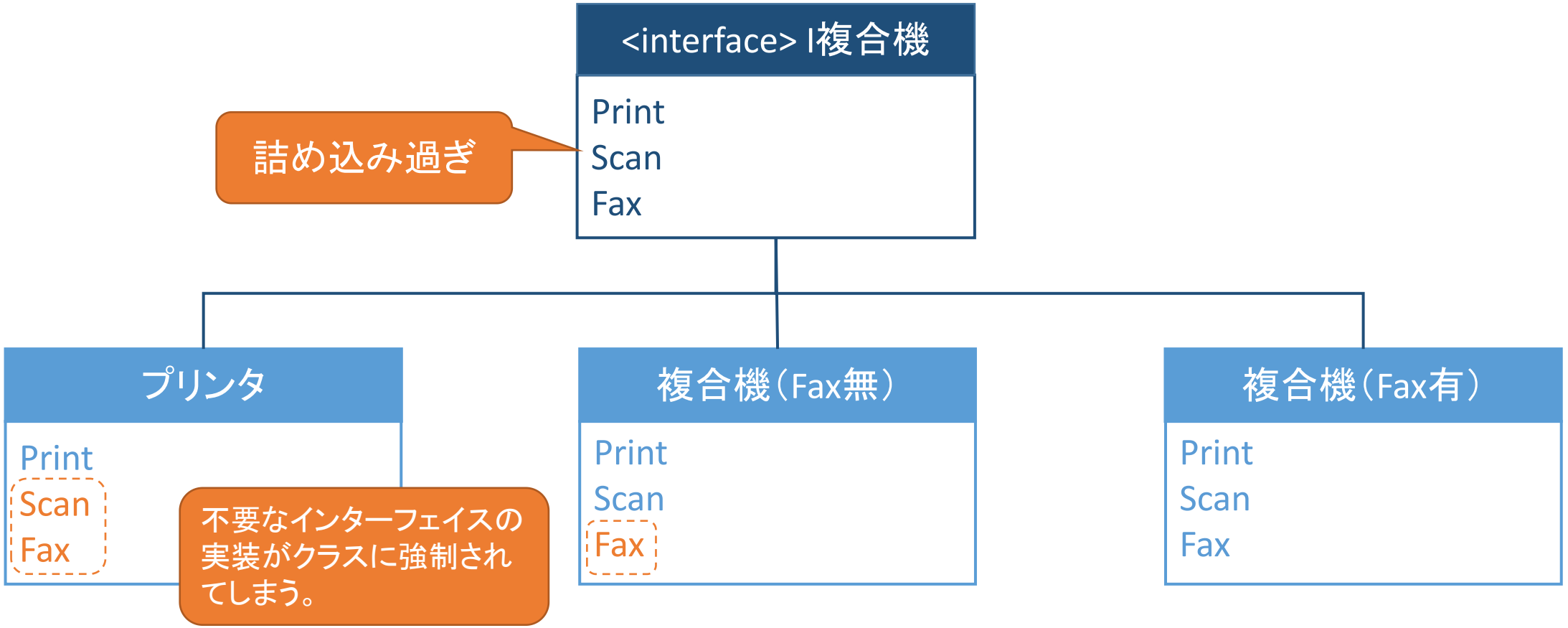
クライアントに、クライアントが利用しないメソッドへの依存を強制してはならない。

解説：

インターフェースや抽象クラスが備えるべきメソッドやプロパティはクライアントにとって必要最小限のものにすべき。クライアントが使用する単位(役割のようなイメージ)でインターフェイスは分けておく必要がある。

オブジェクト指向の5大原則

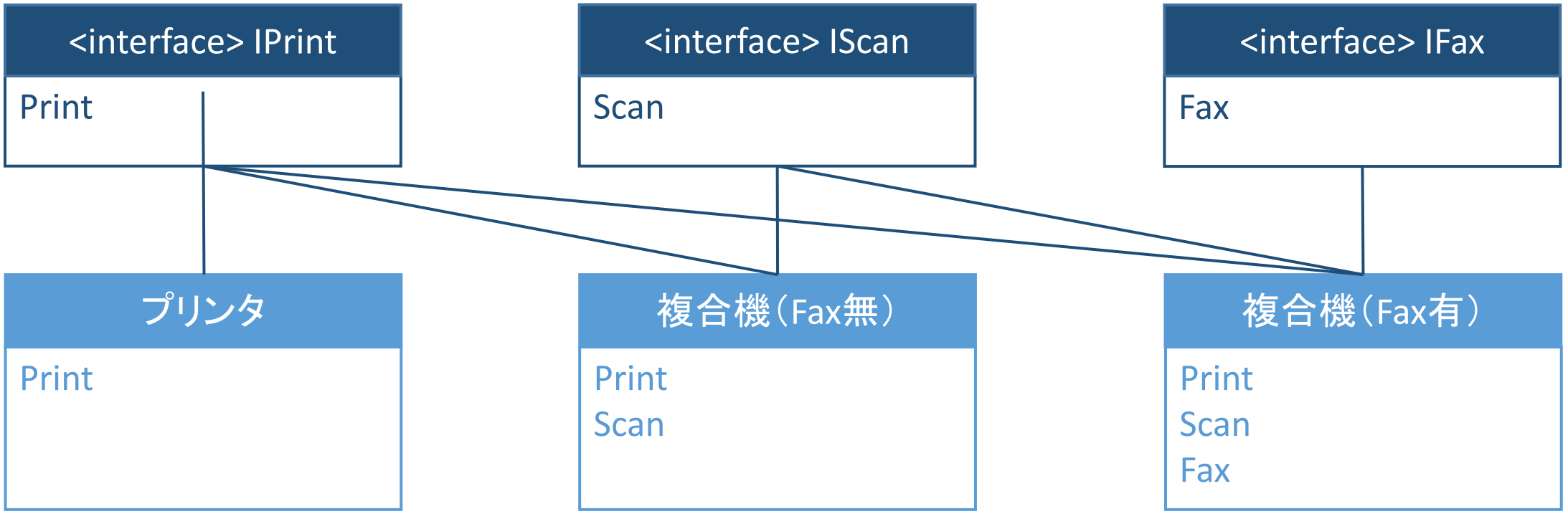
(インターフェイス分離の原則の続き)



オブジェクト指向の5大原則

(インターフェイス分離の原則の続き)

クライアントが使う単位でインターフェイスを分ける。



オブジェクト指向の5大原則

まとめ

- **単一責務の原則** — クラスはなるべく小さく
- **オープンクローズドの原則** — インターフェース（抽象クラス）を使え
- **リスコフの置換原則** — 置換可能でない継承はNG
- **依存関係逆転の原則** — 具象ではなく抽象に依存
- **インターフェイス分離の原則** — 役割としてのインターフェイス（抽象クラス）

デザインパターンとは

- 過去のソフトウェア設計者が構築した設計ノウハウをパターン(型)にしてまとめたもの。
- デザインパターンはオブジェクト指向を有効に使うために作られたものである。
- デザインパターンとしてまとめられているものにもいくつか種類がある。最も有名なものは”GoF”と呼ばれるデザインパターンである。

デザインパターンを学ぶとどうなる？

- デザインパターンにはオブジェクト指向言語をどんな感じでうまく使ってプログラミングしていくかというノウハウが詰まっているため、デザインパターンを学ぶ＝オブジェクト指向の有効な使い方を学ぶということになる。
- パターンをより多く知っておくことで、ソフトウェア設計の現場で多くのパターンの中から最も有効なものを選択できるようになる。