オブジェクト倶楽部 納涼イベント

オブジェクト指向でなぜつくるのか

知っておきたいプログラミング、UML、設計の基礎知識

2004年7月9日

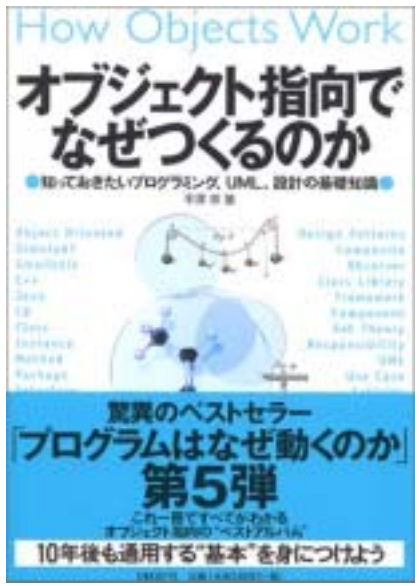
平澤　章

ウルシステムズ株式会社

10年後も通用する“基本”を身につけよう

はじめに

- 「オブジェクト指向でなぜつくるのか」というタイトルの本を書きました。
  - オブジェクト指向ってなんだか…
    - わかったようで、よくわからない。
      - 説明しても通じない。自分でもちゃんと説明できない。
    - 「現実世界とシームレス」の話はどうもインチキ臭い。
  - でも
    - OOPを使ったプログラミングは面白い。
    - UMLによるモデリングは楽しい。
    - デザインパターンを覚えると嬉しい。
  - この10年間で感じてきた、そんな疑問と感覚を振り返り、(私なりに) この技術の正体を明らかにしたいと考えました。



アジェンダ

- 1. オブジェクト指向をめぐる混乱と全体像
- 2. プログラミング技術としてのオブジェクト指向
- 3. 汎用の整理術としてのオブジェクト指向
- 4. UMLモデリングと設計
- 5. おまけ＆まとめ

オブジェクト指向の定義？

- 次の中で、オブジェクト指向の定義として、もっとも適切なものはどれでしょうか？

プログラミング言語は全体の一部

ア. クラス、継承、ポリモーフィズムの機構を備えたプログラミング言語を支える開発・実行環境である。

これは「魔法」

イ. 品質が高く、仕様変更や機能拡張が容易で、大規模に再利用できるソフトウェアを、未熟な要員が劇的な生産性で作れる技術である。

よくある勘違い

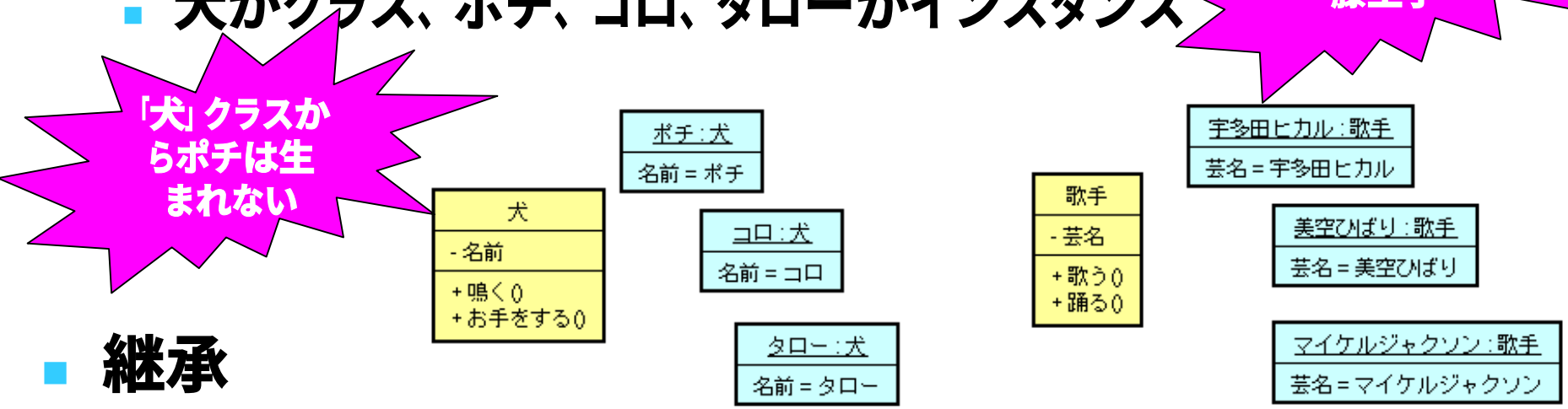
ウ. 森羅万象が「もの」で成り立っている現実世界に対して、「オブジェクト指向」（もの指向）という考え方を適用することで、現実世界をそのままソフトウェアに表現する技術である。

エ. 業務分析から要求定義、設計、プログラミング、開発プロセスまでをカバーするソフトウェア開発の総合的な技術である。

平澤の定義

オブジェクト指向と現実世界は大違い(1)

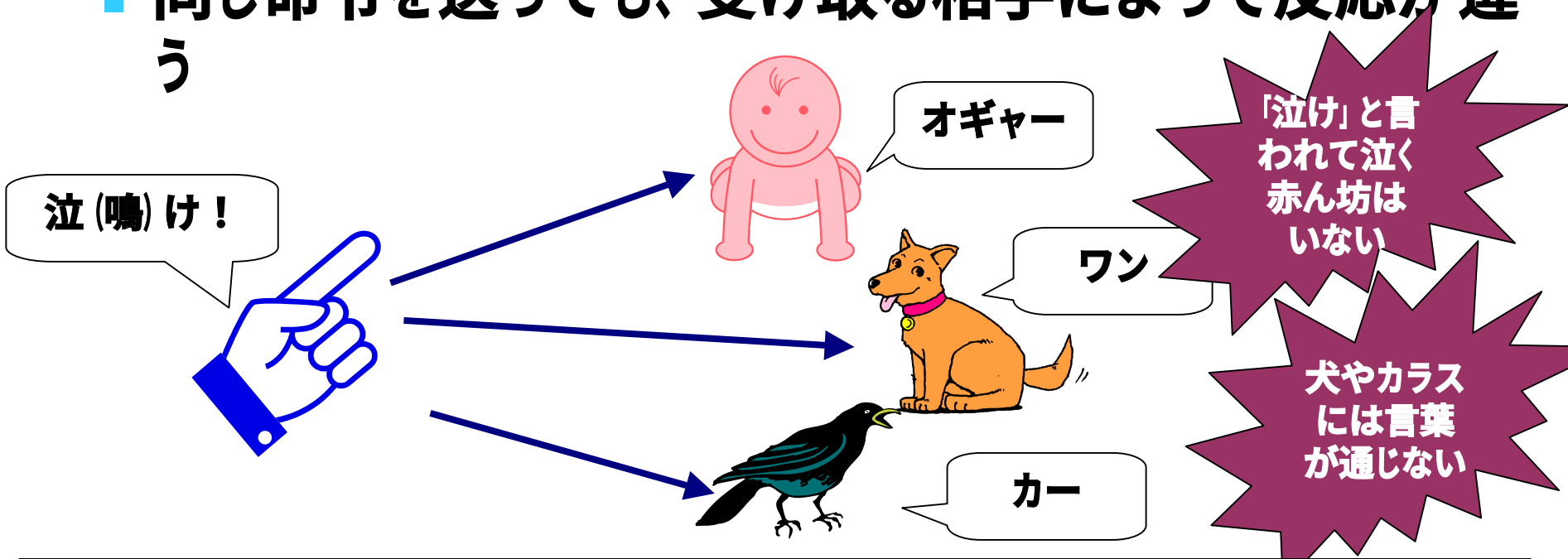
- クラスとインスタンス
  - 犬がクラス、ポチ、コロ、タローがインスタンス



- 継承
  - 「男」のサブクラスは「赤ん坊」「少年」「青年」「中年」「熟年」

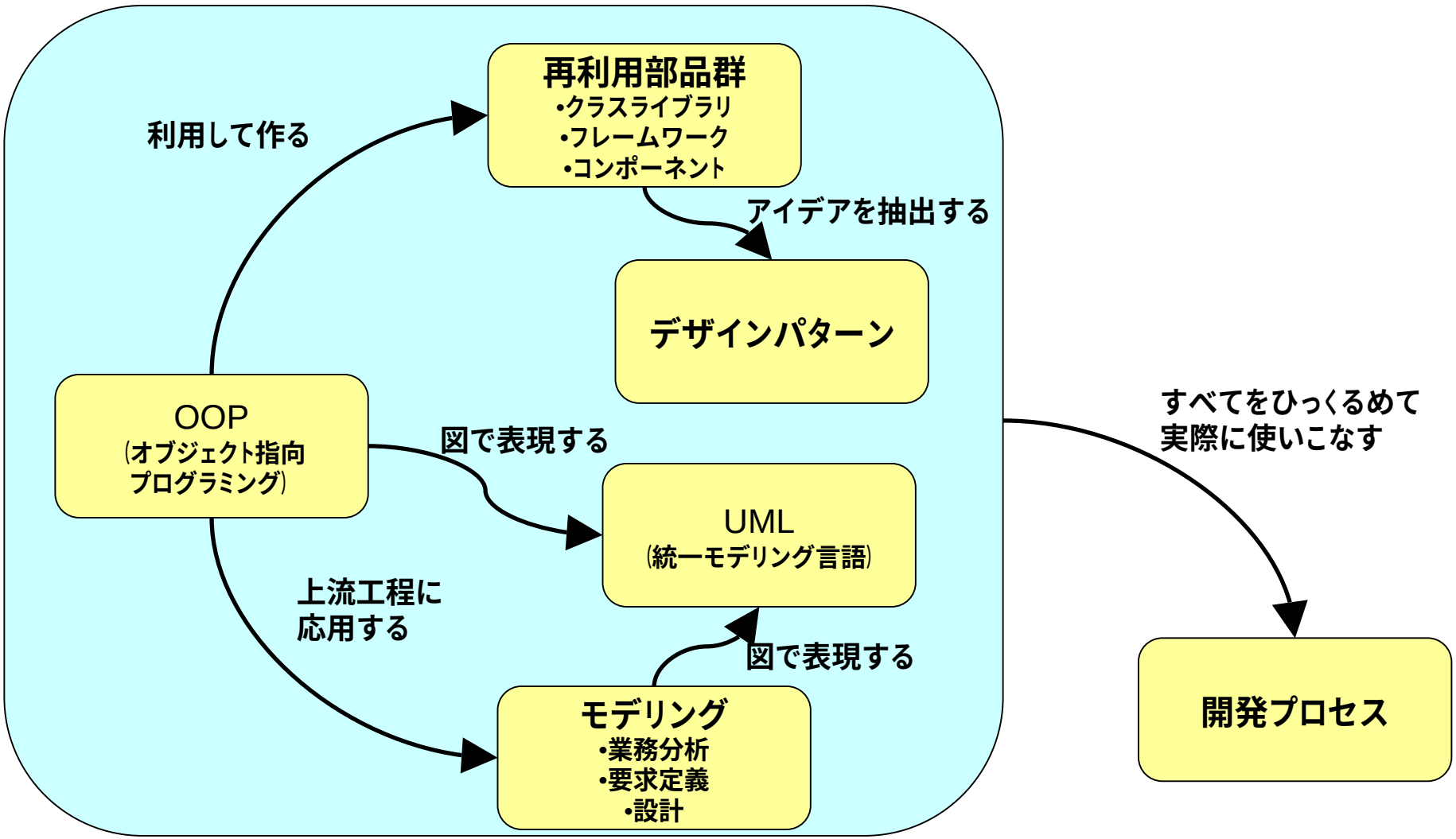
オブジェクト指向と現実世界は大違い(2)

- ポリモーフィズム
  - 同じ命令を送っても、受け取る相手によって反応が違う



こうした説明で混乱しないコツは、
あくまでイメージをつかむための「たとえ話」と割り切ること！

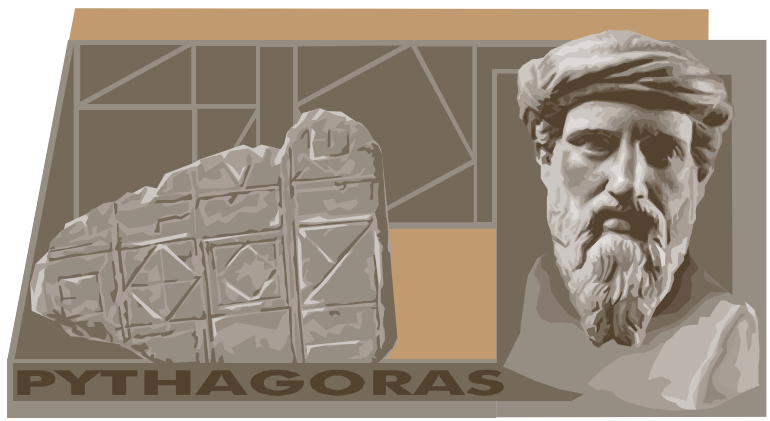
プログラミング言語から発展した総合技術



「プログラミング技術」と「汎用の整理術」に分けて理解しよう

- プログラミング技術
  - 優れたプログラミング言語
    - Java, C#, C++, Smalltalk, Ruby, …
    - クラス、ポリモーフィズム、継承の3つの仕組み
  - そこから発展した再利用技術
    - クラスライブラリ、フレームワーク、コンポーネント
    - アイデアの再利用としてのデザインパターン

- 汎用の整理術
  - 集合論、役割分担
    - 業務分析、要求定義への応用
    - 整理した結果を図式表現するUML
    - 哲学や認識論までをカバー!?

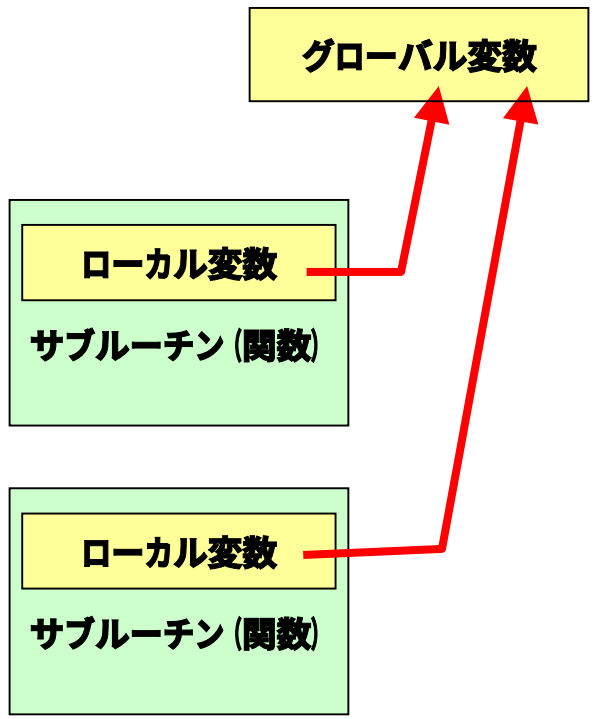


アジェンダ

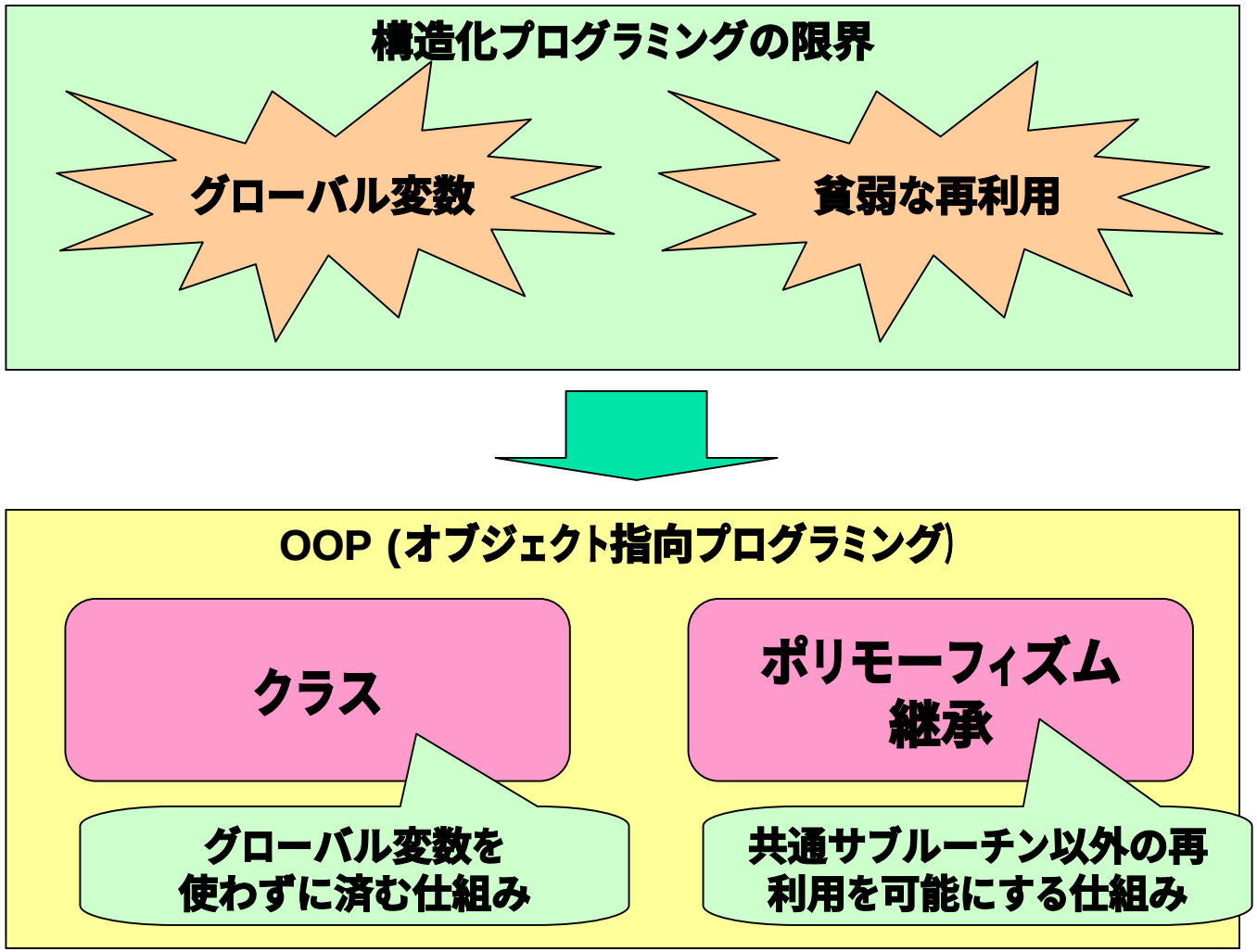
- 1. オブジェクト指向をめぐる混乱と全体像
- 2. プログラミング技術としてのオブジェクト指向
- 3. 汎用の整理術としてのオブジェクト指向
- 4. UMLモデリングと設計
- 5.おまけ＆まとめ

OOP前夜 - 構造化プログラミング

- プログラムの構成要素
  - サブルーチン (関数)
  - グローバル変数、ローカル変数
- プログラミングの作法
  - 基本三構造
    (GOTOレスプログラミング)
    - 順次進行
    - 条件分岐 (if, case)
    - 繰り返し (for, while)
  - サブルーチン (関数) の独立性確保



構造化プログラミングの限界

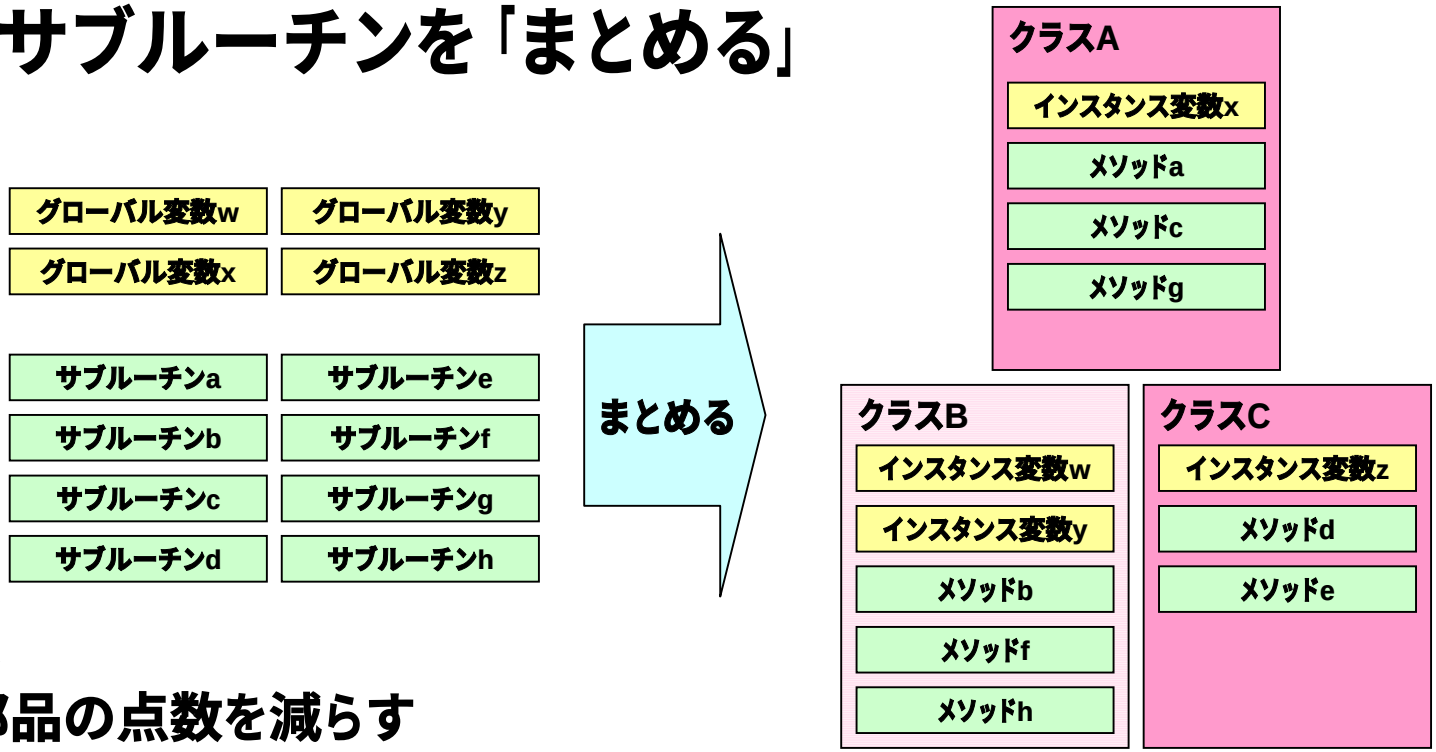


OOPの3つの仕組み

- クラス
  - 「まとめて」「隠して」「たくさん作る」仕組み
    - 変数とサブルーチンを「まとめる」
    - クラスの内部だけで使う変数やサブルーチンを「隠す」
    - 1つのクラスからインスタンスを「たくさん作る」
- ポリモーフィズム　(インタフェースの継承)
  - 「共通メインルーチン」を作る仕組み
- 継承　(実装の継承)
  - クラス定義の共通部分を別クラスにまとめる仕組み

クラスの効能1 – まとめる

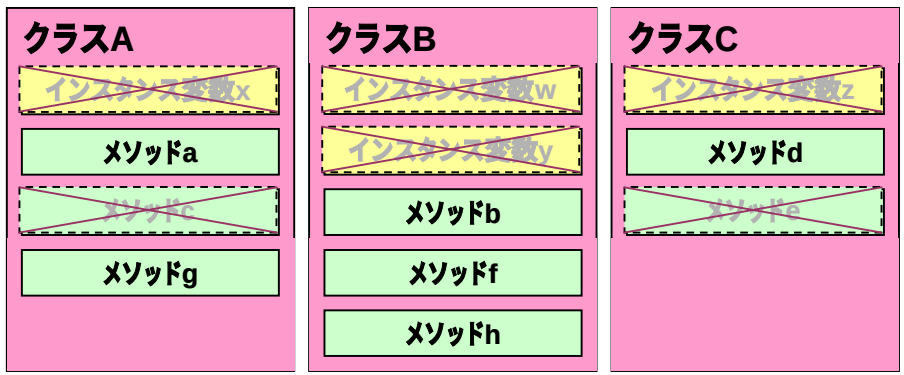
- 変数とサブルーチンを「まとめる」



- 効果
  - 部品の点数を減らす
  - サブルーチンや変数を探しやすくする
  - 名前づけを楽にする
  - まとめて「型」として取り扱うことができる

クラスの効能2 – 隠す

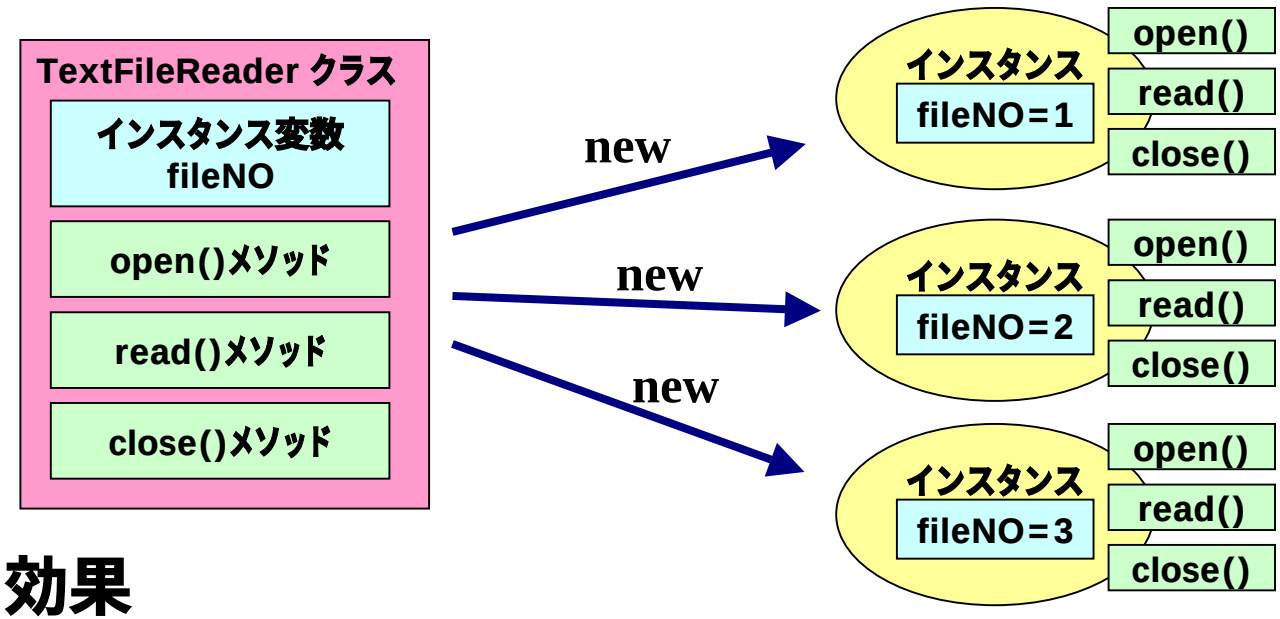
- クラスの内部だけで使う変数やサブルーチンを「隠す」



  - 効果
    - 修正時の影響範囲を小さくする。
    - 見通しを良くする。

クラスの効能3 – たくさん作る

- 1つのクラスからインスタンスを「たくさん作る」



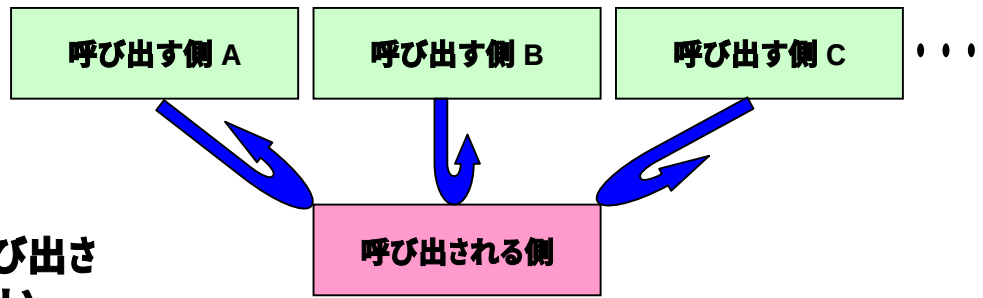
- 効果
  - 同種の情報を複数同時に扱うロジックを単純化できる。

「インスタンス変数名.メソッド名(引数)」のように、インスタンス変数を指定してメソッドを呼び出す理由は、処理対象とするインスタンスを特定するため。

例) fileReader.open ("C:¥tmp¥TestFile1.txt");

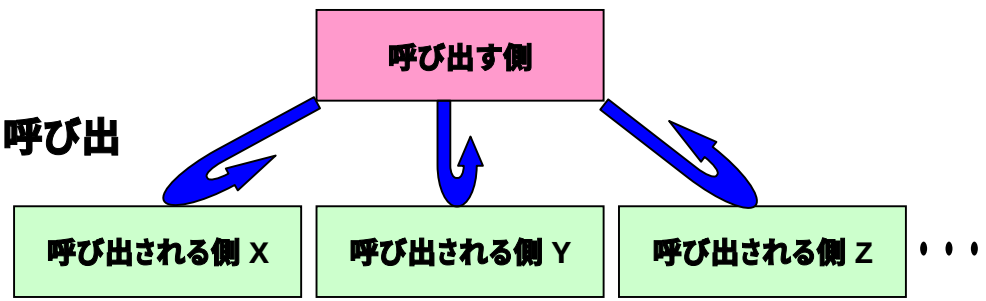
ポリモーフィズム

- 呼び出し側を共通化する、
  すなわち「共通メインルーチン」を作る仕組み



共通サブルーチン

呼び出す側が増えても、呼び出される側を修正する必要がない

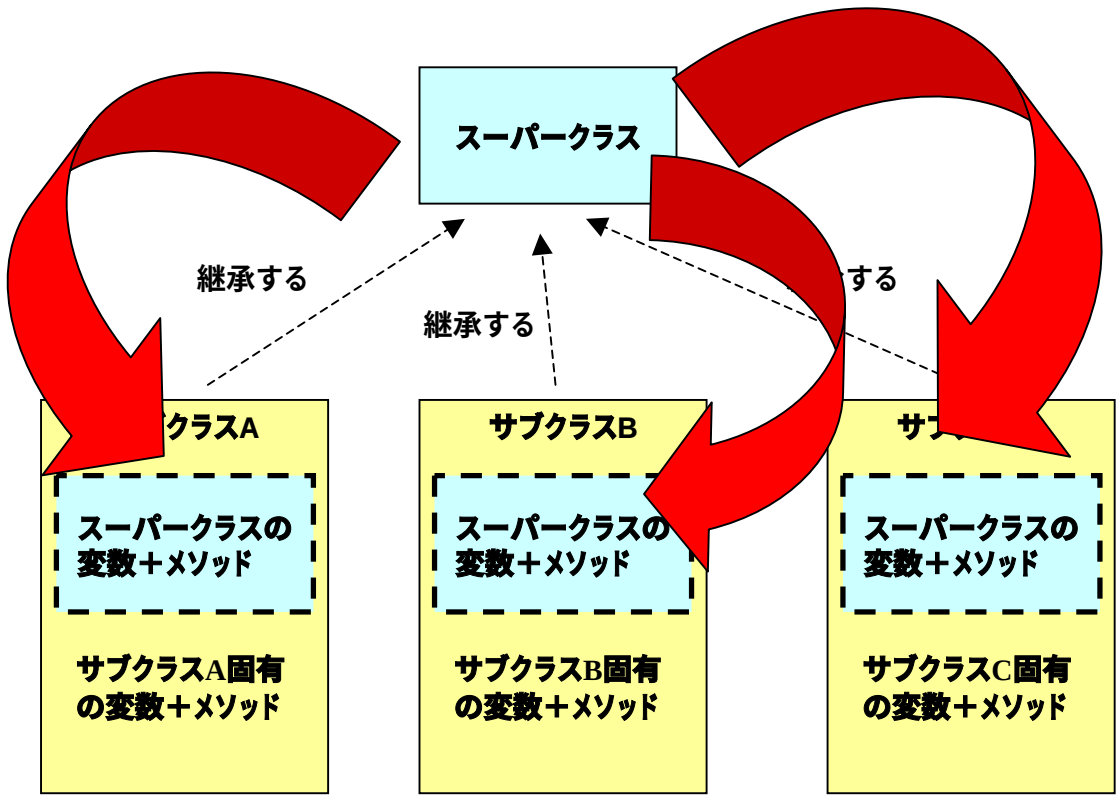


ポリモーフィズム

呼び出される側が増えても、呼び出す側を修正する必要がない

継承

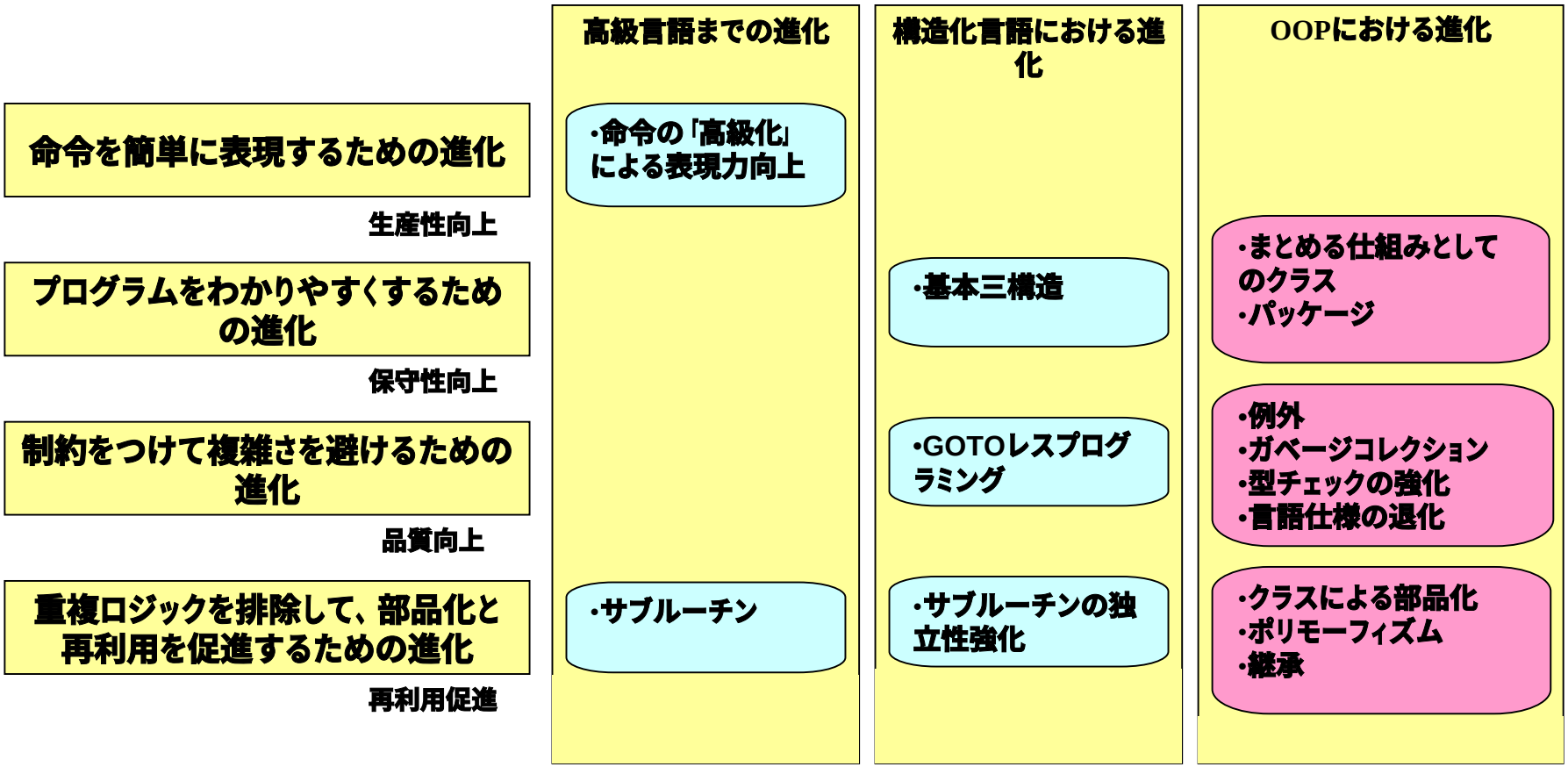
- クラス定義の共通部分を別クラスにまとめる仕組み



さらに進化したOOPの仕組み

- パッケージ
  - クラスをさらに「まとめる」仕組み
- 例外
  - 戻り値とは違う方式でエラー処理をする仕組み
- ガベージコレクション
  - 不要になったインスタンスを自動的に削除する仕組み
- その他
  - 表明、テンプレート、リフレクション…

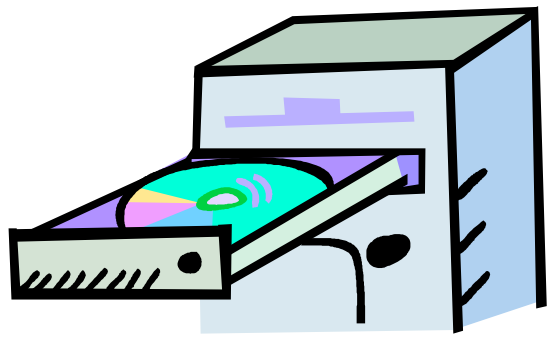
OOPは構造化プログラミングの発展形



OOPは連綿と続くプログラミング言語の工夫と改良の歴史の中で、必然性を持って登場した技術

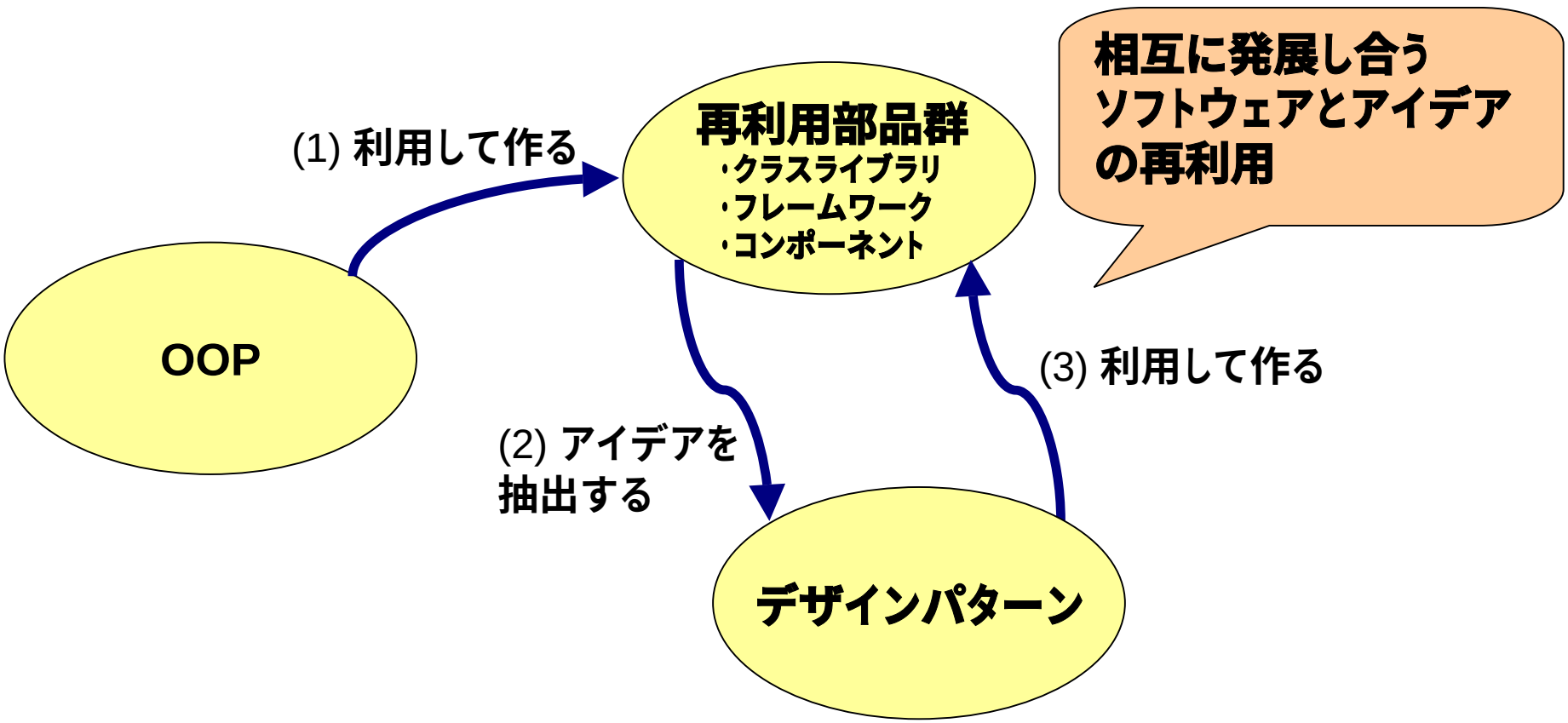
OOPがもたらした2つの再利用

- ソフトウェアそのものの再利用
  - クラスライブラリ
  - フレームワーク
  - コンポーネント

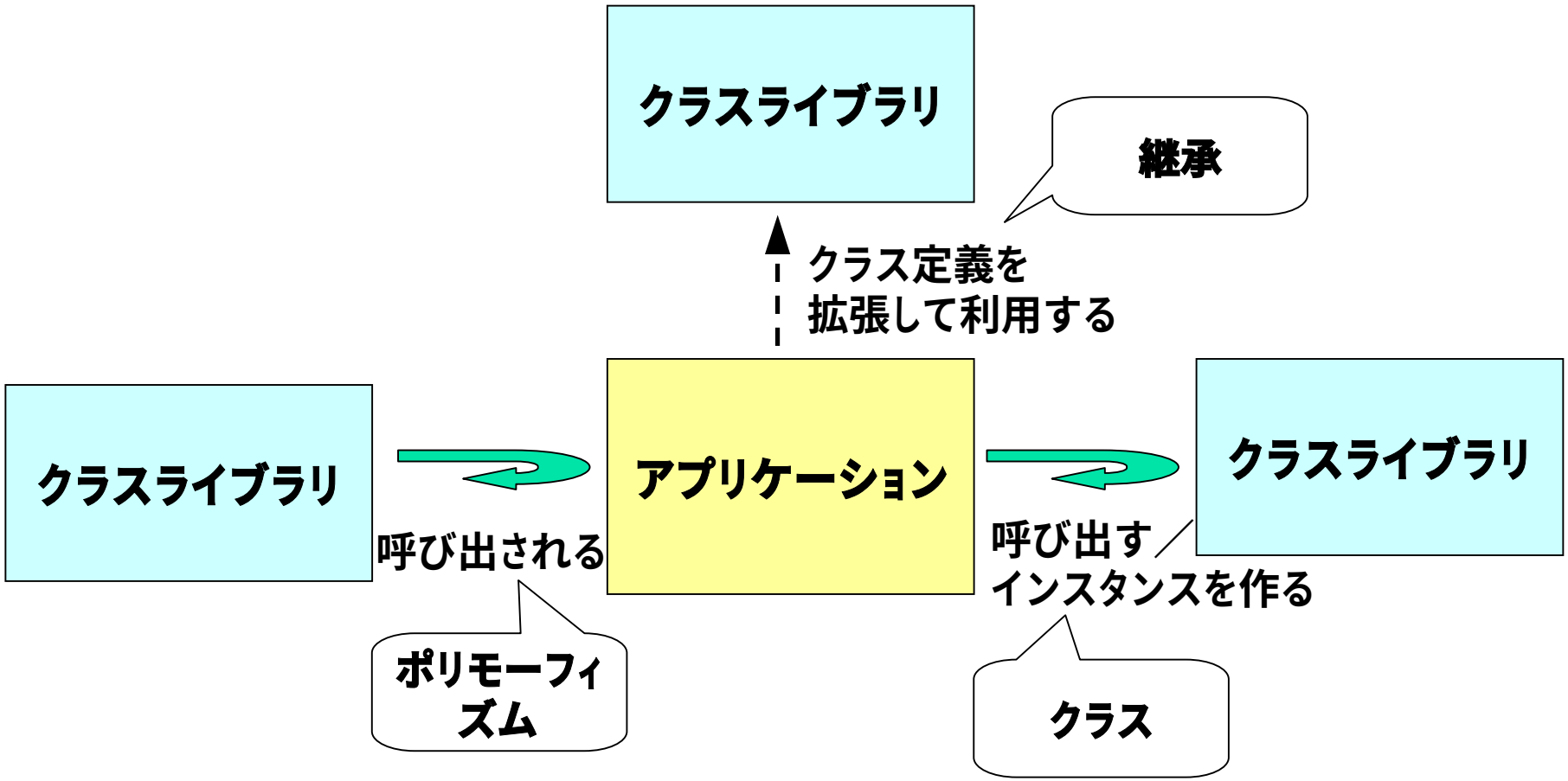


- アイデアの再利用
  - デザインパターン

OOPがもたらした2つの再利用



OOPが大きく広げたソフトウェアの再利用



アジェンダ

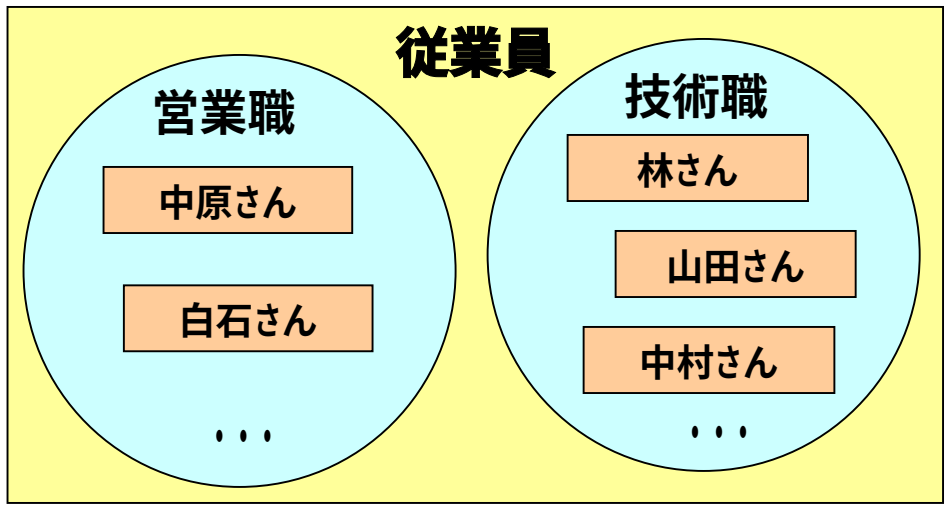
- 1. オブジェクト指向をめぐる混乱と全体像
- 2. プログラミング技術としてのオブジェクト指向
- 3. 汎用の整理術としてのオブジェクト指向
- 4. UMLモデリングと設計
- 5. おまけ＆まとめ

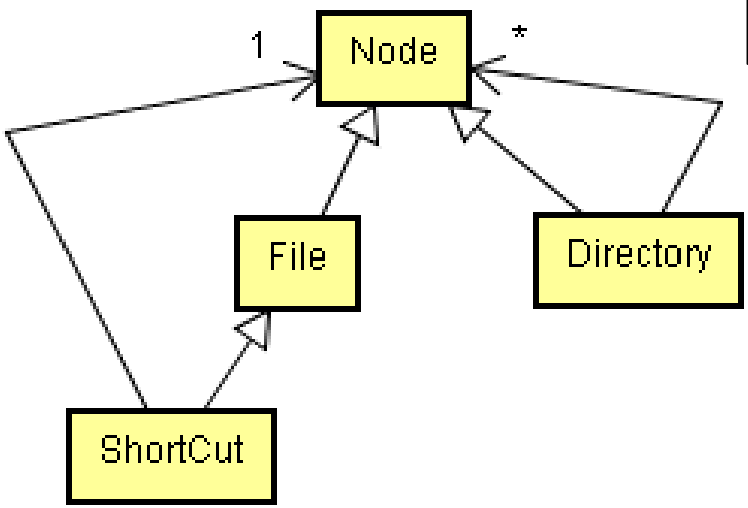
上流工程で汎用の整理術に化けた

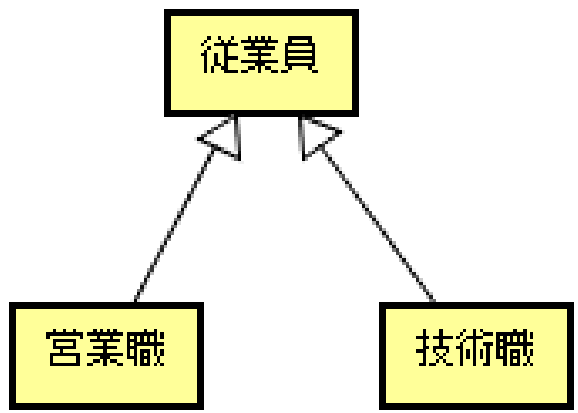
- オブジェクト指向は上流工程に応用された
  - OOPは、優れたプログラミング技術として保守性や再利用性の高さが認められた。
  - 次の段階では、ソフトウェア開発全体の生産性を上げるために、上流工程に応用された。

クラスは集合論に

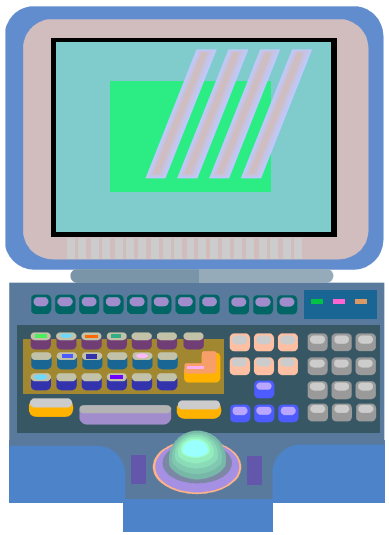
```
public abstract class Node {
    private String name;
    protected Node(String name) {
        this.name = name;
    }
    public String getName() {
        return this.name;
    }
    public void setName(String name) {
        this.name = name;
    }
```

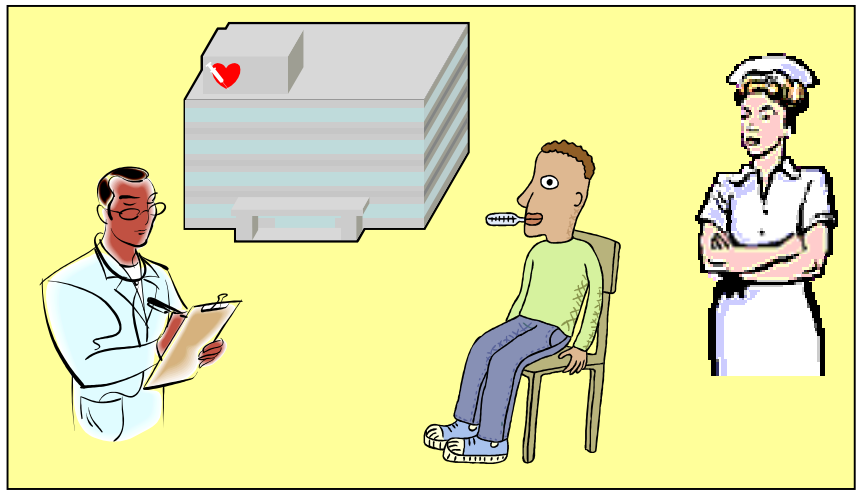


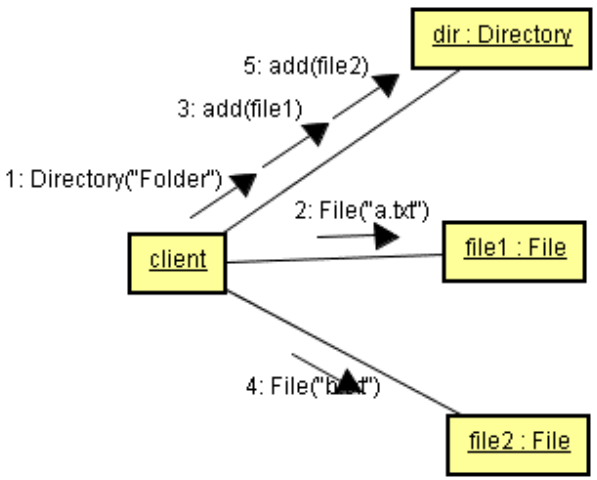


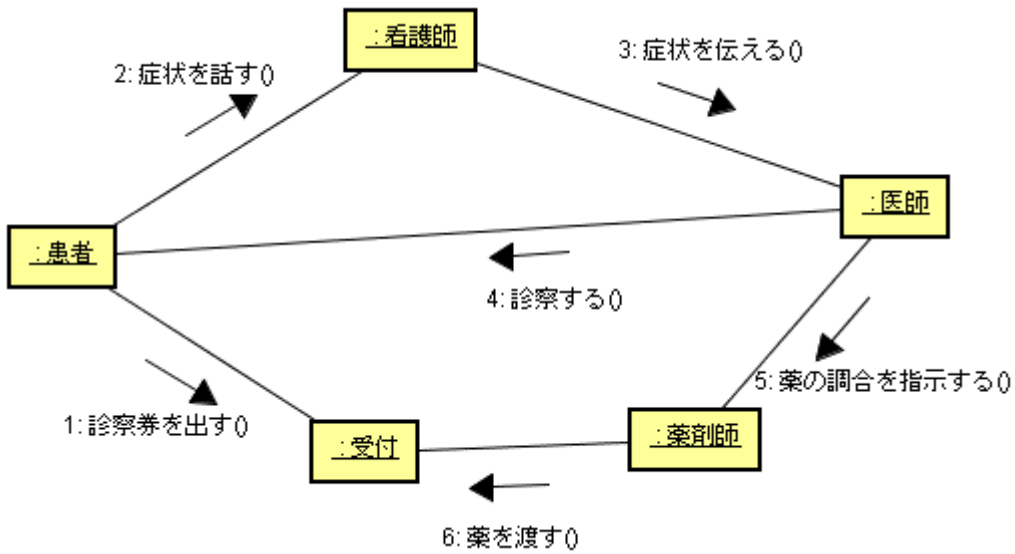


メッセージパッシングは役割分担モデルに









2つの意味を持つことが混乱をもたらした

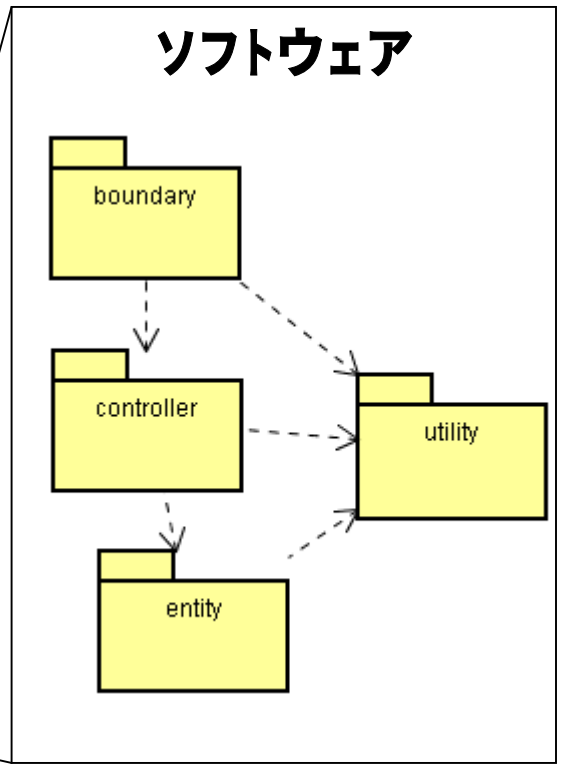
- クラス
  - プログラミング技術
    - サブルーチン (関数) と変数をまとめる仕組み
  - 汎用の整理術
    - 森羅万象を表現する「ものの種類」
- メッセージパッシング
  - プログラミング技術
    - 「たくさん作られた」インスタンスの1つを指定して、メソッドを呼び出す仕組み
  - 汎用の整理術
    - 組織や人が協調して全体の仕事を達成する様子を表現するモデル

プログラミング技術と汎用の整理術は「似て非なるもの」と割り切るのが混乱を避けるコツ

アジェンダ

- 1. オブジェクト指向をめぐる混乱と全体像
- 2. プログラミング技術としてのオブジェクト指向
- 3. 汎用の整理術としてのオブジェクト指向
- 4. UMLモデリングと設計
- 5. おまけ＆まとめ

現実世界とソフトウェアのギャップを埋める3つのステップ



業務分析

現実世界の仕事の様子を整理する

要求定義

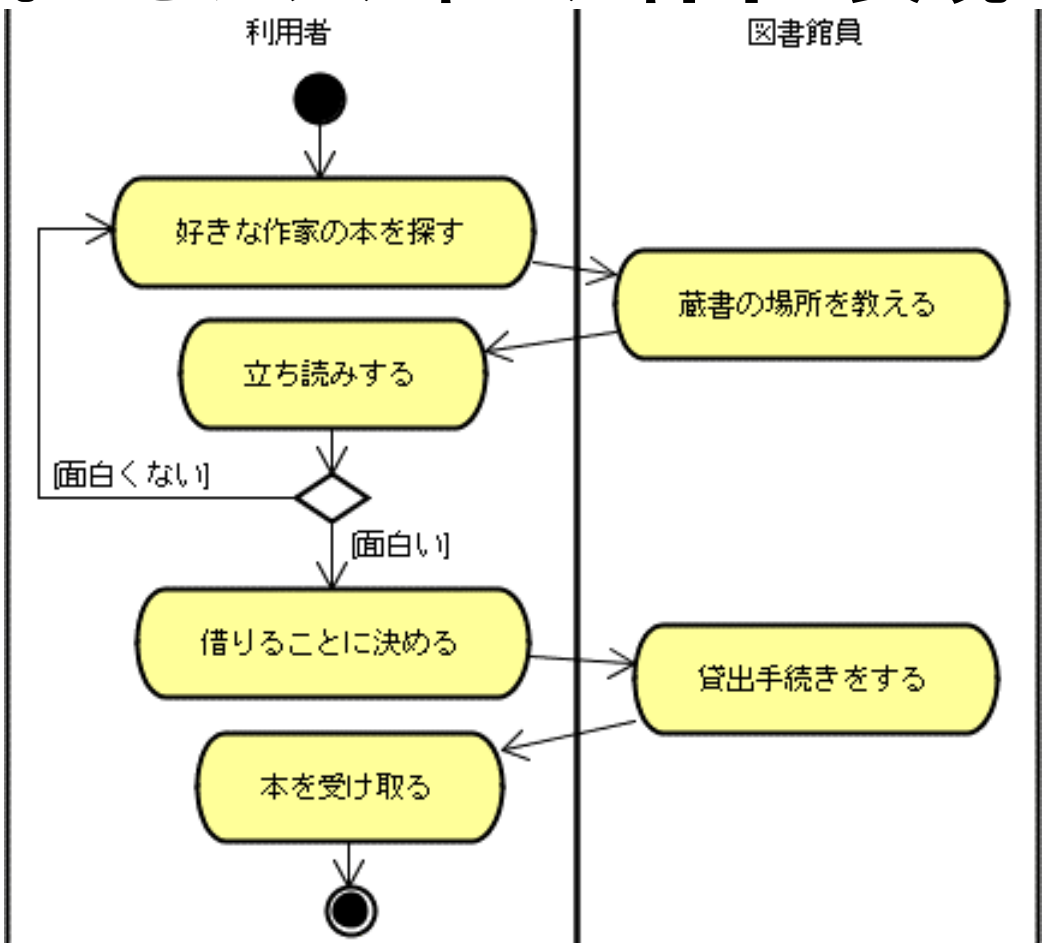
コンピュータにまかせる仕事の範囲を決める

設計

ソフトウェアの構造を決める

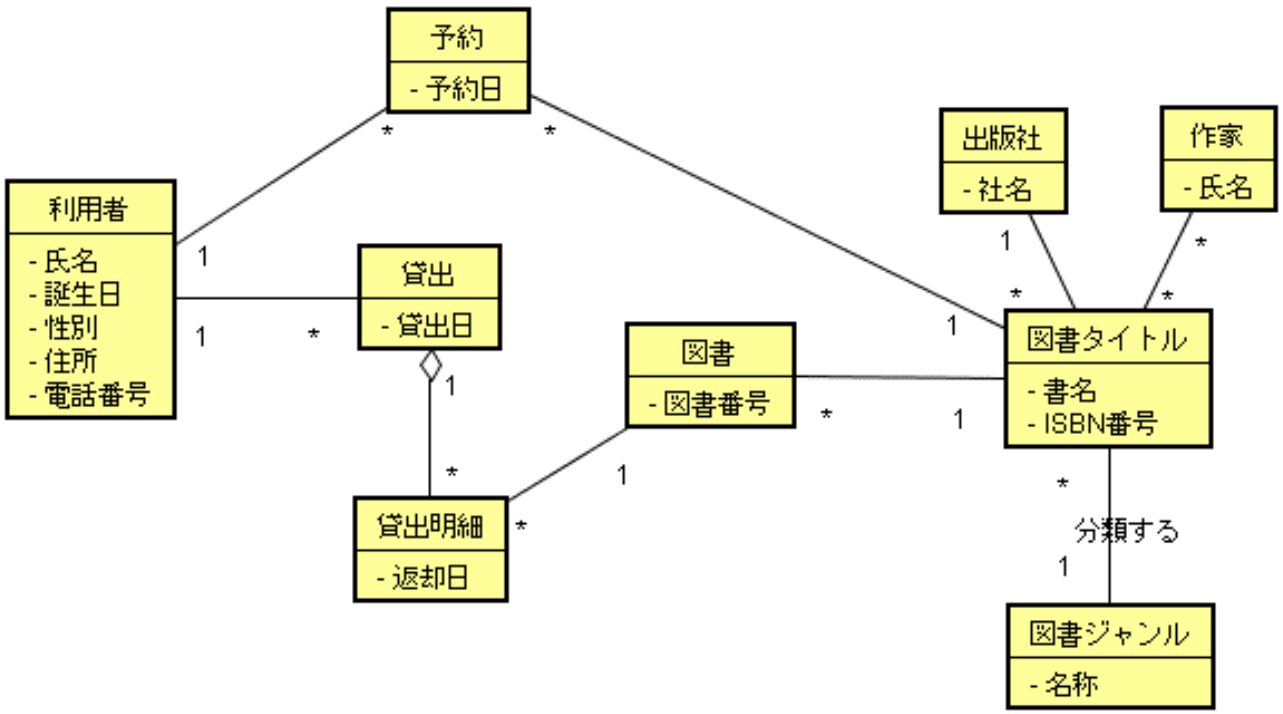
業務分析

- 業務の流れをアクティビティ図で表現



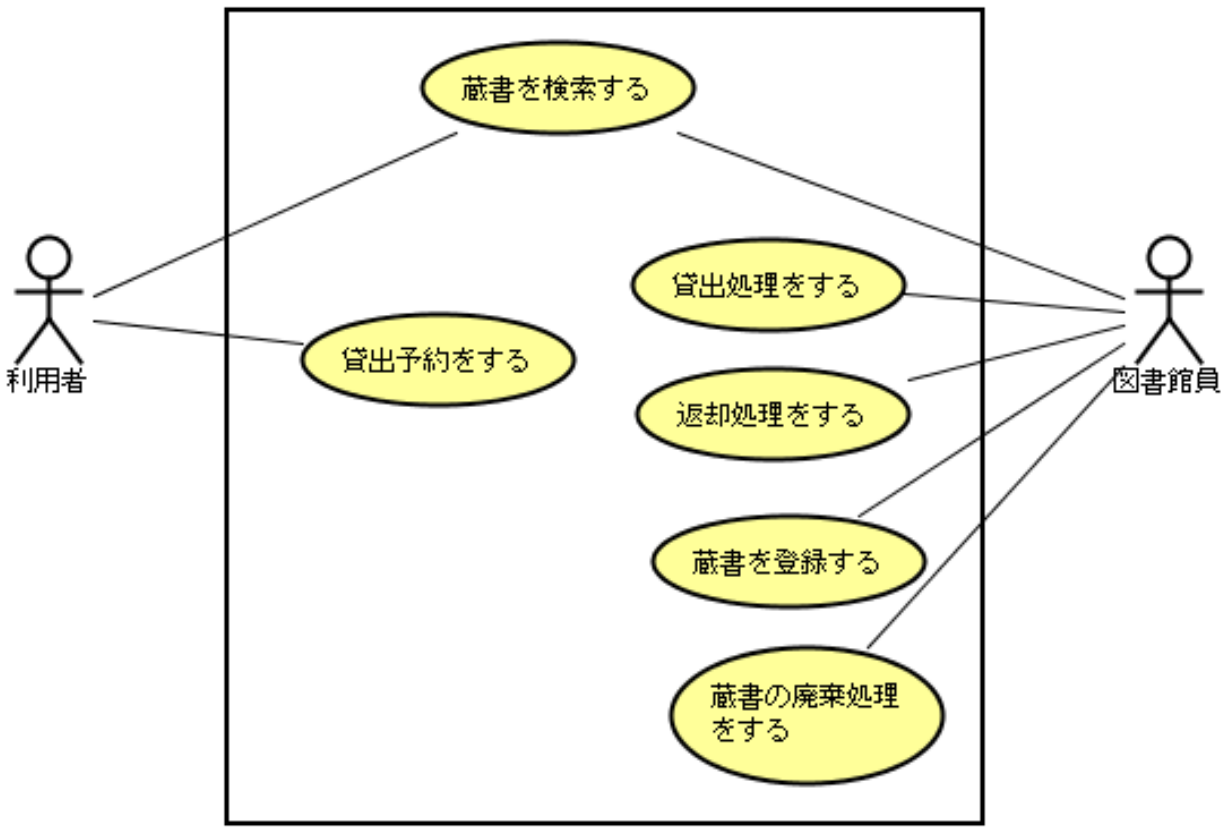
要求定義 – 概念モデリング

- 管理する情報の構造をクラス図で表現



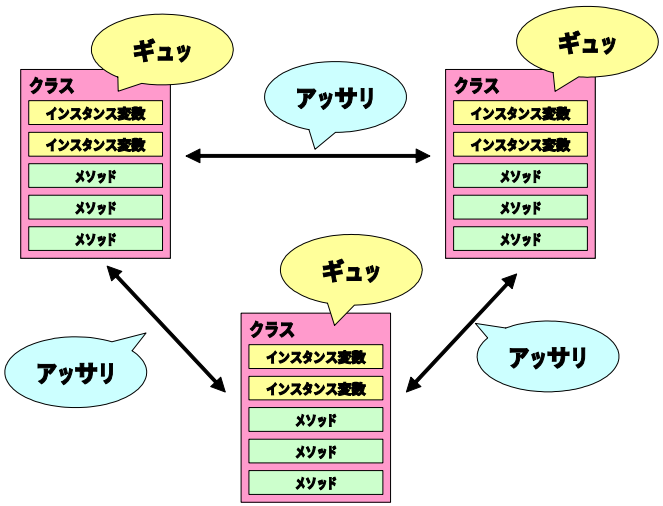
要求定義 – ユースケース定義

- 提供するサービスをユースケース図で表現

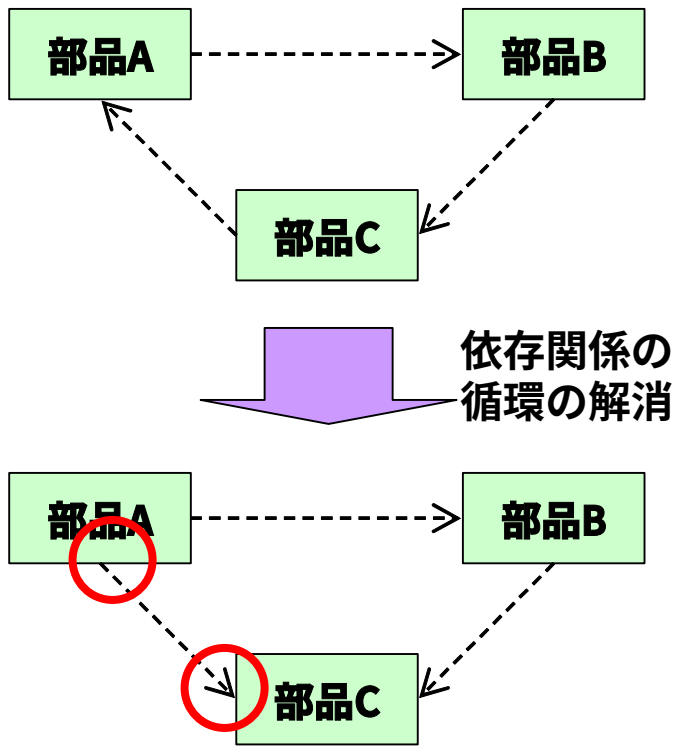


オブジェクト指向設計

- 保守性と再利用性を向上させるための目標
  - 1. 重複の排除
  - 2. 部品の独立性確保
  - 3. 依存関係の循環の回避

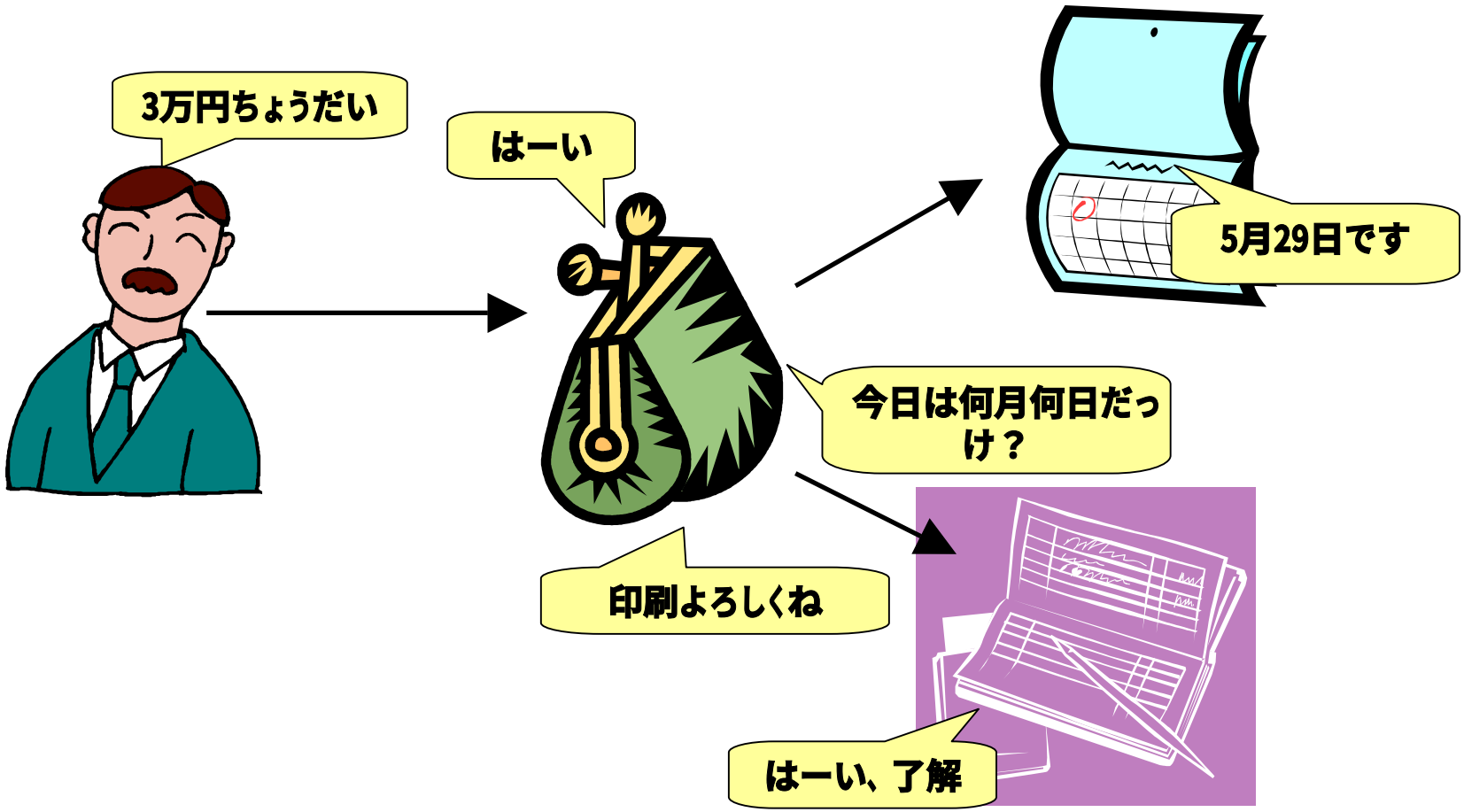


部品の独立性確保

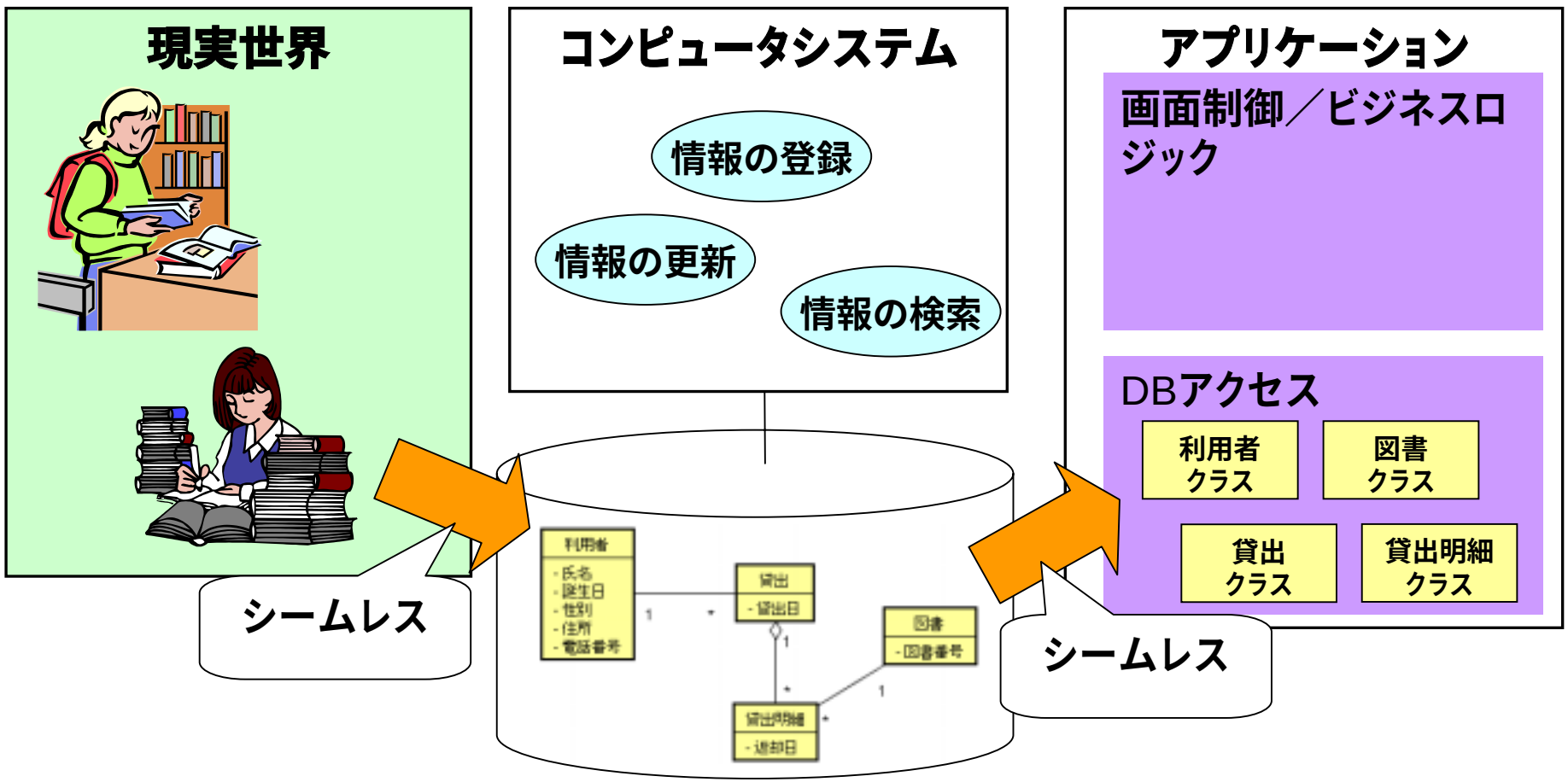


オブジェクト指向設計のコツ

- 命のないソフトウェアを擬人化して役割分担する



シームレスなのはデータだけ



アジェンダ

- 1. オブジェクト指向をめぐる混乱と全体像
- 2. プログラミング技術としてのオブジェクト指向
- 3. 汎用の整理術としてのオブジェクト指向
- 4. UMLモデリングと設計
- 5. おまけ＆まとめ

オブジェクト指向 - 最も成功したバズワード?

- Smalltalkを開発したアラン･ケイ氏が加えた強烈なコンセプト
  - Object クラスを最上位クラスとした継承構造
    - すべてはオブジェクト Everything is an Object!
      - すべてのクラスはObjectクラス (のサブクラス)
      - すべてが「もの」から成り立っている現実世界
- ソフトウェア業界でもっとも成功したバズワード？
  - 抜群のセンス
    - EA, SOAなんて目じゃない
  - 広がりがある
    - プログラミングから、集合論、哲学までをカバー
    - 設計だけでなく、要求定義、業務分析までが対象範囲
  - 発想の転換
    - 機能中心の考え方とは違う！

5. おまけ＆まとめ

書籍「オブジェクト指向でなぜつくるのか」の登場人物

- ソフトウェア技術者
  - O.J. ダール氏と K. ニガード氏 (2001年チューリング賞)
    - http://www.ifi.uio.no/adminf/tribute.html
  - アラン・ケイ氏 (2003年チューリング賞)
    - http://www.squeakland.org/images/news/html/turing03.htm
  - スリーアミーゴ
  - ギャング・オブ・フォー

- ミュージシャン
  - ピンク・レディー (p17 第1章クイズ)
    - http://www.pinklady.org/
  - ビートルズ (p18 第1章クイズ)
    - http://www.beatles.com/
  - エマーソン、レイク＆パーマー (p152 メモリ配置)
    - http://www.emersonlakepalmer.com/
  - スリードッグナイト (p199 第8章クイズ)
    - http://www.threedognight.com/
  - スリーディグリーズ (p199 第8章クイズ)
    - http://www.eonet.ne.jp/~makochan/newpage11.htm

オブジェクト指向でなぜつくるのか

- 現実世界をそのままプログラムに表現する技術**ではない**。
- ソフトウェア開発を楽にする総合技術
  - 優れたプログラミング言語＋再利用部品群
    - 実績のある再利用部品を使い回せる
    - あとから保守しやすく作る仕組みを備えている
  - デザインパターン
    - 先人のノウハウを活用できる
  - UMLによるモデリング
    - 何もないところから、ユーザーの要求を形にできる
  - アジャイル開発プロセス
    - メンバーのやる気を引き出し、顧客と協調してシステムを開発できる

オブジェクト指向を使いこなして、
知的なソフトウェア開発を楽しもう！