プログラミング作法

第一回　入門

プログラミング作法とは？

プログラムの文法に加えて，プログラムを書く上で重要な考え方や書き方を学ぶことでより良いプログラムを作ることができるようになる．

◆作法を知ることの利点

■バグを埋め込みにくくなる

- バグは新たなバグを生む原因になる
- （実験においても）バグは結果の違いを生み致命傷になるかも

■コードを読み込みやすくなる

- コードを読むのは自分だけだとは限らない
- （実験においても）他の人が理解するのに苦労するかも

■コードを変更しやすくなる

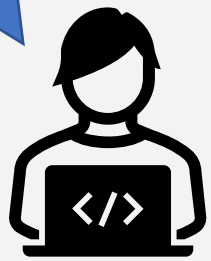
- 機能を追加しやすいプログラムは成長しやすいプログラム
- （実験においても）追加実験がやりやすくなるかも

なんで今学ぶの？

良いコードの書き方はいずれ学ぶ必要がある（プログラマーになるなら）し，
研究でも役に立つ知識はたくさんある．

作法を知らないと…

文法は覚えたし
とにかく書こう！

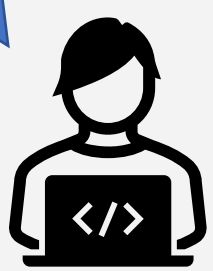


過去の自分

```
public class Test001 {
private String name;
private int long;
…
}
```

次の日

あれ，これ何だっけ…
（また読み直さなきゃ…）



現在の自分

```
public class Test001 {
private String name;
private int long;
…
}
```

Nameって何の
名前だよ…

具体的な内容は（1/2）？

「良いコード/悪いコードで学ぶ設計入門（著：仙塲大也）」の内容に沿って学ぶ.

本書を選ぶ理由

- 2022/4/30出版と比較的新しい
  （リーダブルコードは2012/6/23）
- Javaで書かれている
  （Javaは設計の話題が豊富）
- 悪い例から学ぶことで何がより良いコードか学びやすい
  （悪いコードを避けることが大切）

出典：https://amzn.asia/d/fHypO5e

具体的な内容は（2/2）？

「良いコード/悪いコードで学ぶ設計入門（著：仙塲大也）」の内容に沿って学ぶ.

■取り扱う内容（予定）

1. 悪しき構造の弊害
2. 設計の初歩
3. クラス設計
4. 不変の活用
5. 低凝集
6. 条件分岐
7. コレクション
8. 密結合
9. 設計の健全性
10. 名前設計
11. コメント
12. メソッド・関数
13. モデリング
14. リファクタリング

入門

実践

発展

出典：https://amzn.asia/d/fHypO5e

え，Java知らないんですけど…

大丈夫です．他の言語にも通じる話なので…

◆現在の人気NO.1の言語はPython

じゃあなんでJavaで紹介されているの？

◆現場ではまだまだJavaの需要が高く，情報も多い（実際求人が一番多いのはJava）

◆C#などのオブジェクト指向を取り入れた言語を知っているなら問題無く学習できる.

◆Javaもしくはオブジェクト指向について詳しく知りたい方は別の勉強会を開くことを考え中です.（ラムダ式，コレクション，並列処理etc.）

Javaとオブジェクト指向

ちょっとだけ説明しておきます.

クラスの概念が理解できればとりあえず大丈夫！

JavaとPythonの違い

JavaとPythonは主に以下のような違いがある.

◆Java

- ■需要No.1
- ■オブジェクト指向
- ■静的型付け
- ■学習が少し難しい…
- ■汎用性が高い！
- ■その他文法的な違い

```
public class Mian {
    public static
     void main(String args[])  {
    System.out.print(“Test”);
    }
}
```

◆Python

- ■人気No.1
- ■手続き型（オブジェクト指向も可能）
- ■動的型付け
- ■学習が簡単！
- ■ライブラリが豊富！

```
print(Test)
```

※左右のコードは同じ内容

オブジェクト指向とは？

オブジェクト指向とは，プログラムを書く際に用いる部品化の考えのこと.

クラス，表明，総称性，継承，多相性，動的束縛性から構成されるもの.

※1

※1　『オブジェクト指向入門 第二版 原則・コンセプト』より

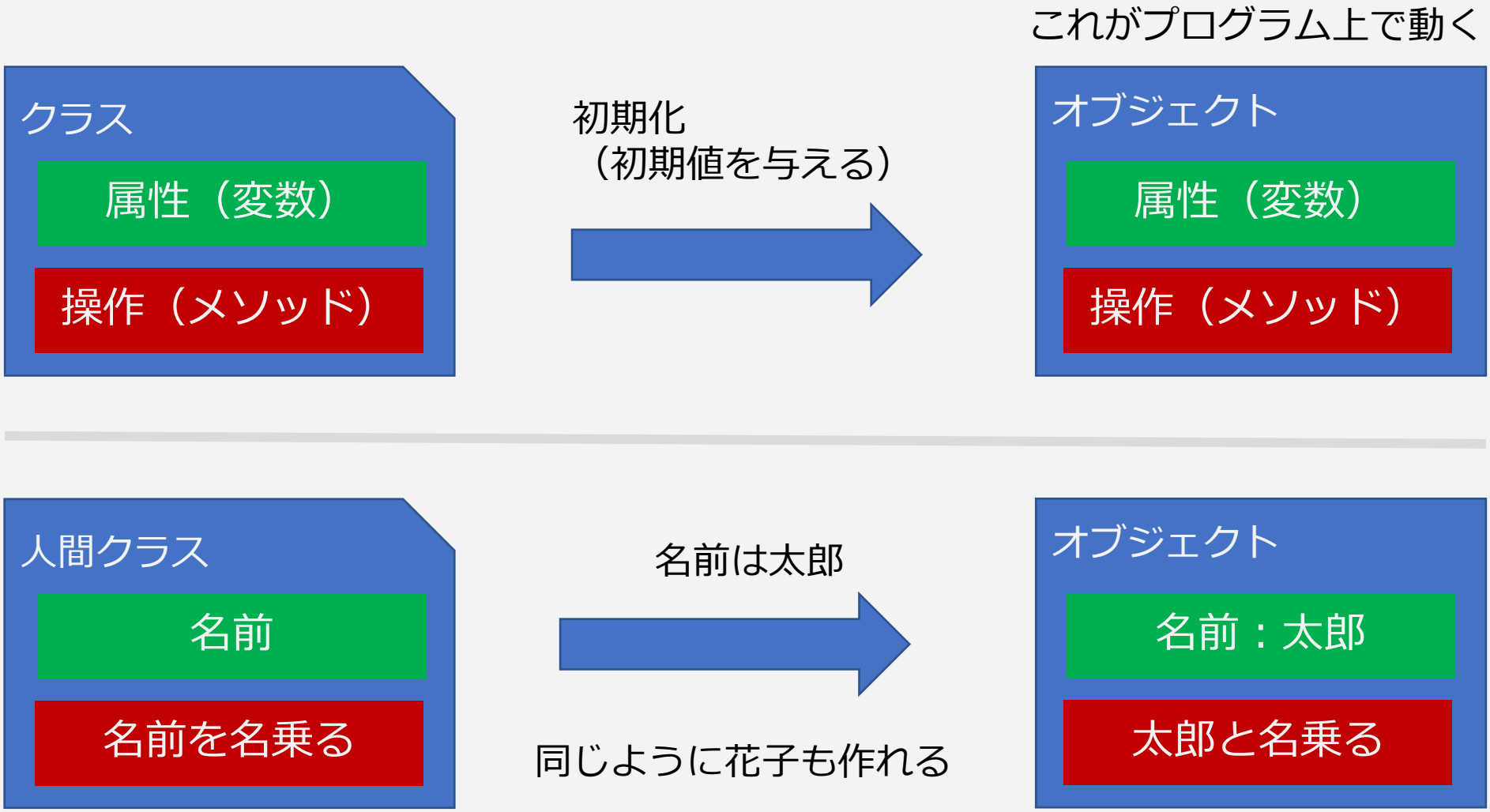
（簡単に言えば）プログラムを書く際に用いる部品化の考えのこと.

※2

※2　『スッキリ分かる．Java入門 第三弾』より

（説明に入る前に）クラスってなに？

クラスはオブジェクト（インスタンス）の設計図の様なもので，属性と操作を持つ.



オブジェクト指向と手続き型の違い

ゲームの世界における戦闘で例えてみる.

◆手続き型だと…

■指示１
- 主人公が攻撃

■指示２
- 敵AのHPを５減らす

■指示３
- 敵のHPが０なら戦闘終了

■指示４
- 敵が攻撃

■指示５

…（以下戦闘が終わるまで指示を続ける）

◆オブジェクト指向なら

■主人公を生成
- 敵に攻撃したらダメージを与える
- HPが０以下になったら戦闘終了

■敵を生成
- 主人公に攻撃したらダメージを与える
- HPが０以下になったら戦闘終了

■戦闘を生成

■実行（あとは見るだけ）

ざっくりとしたJavaの世界(1/2)

例えば実験でJavaを使うならこんな感じ.

JVM：Java Virtual Machine （仮想世界）

Mainオブジェクト

Mainメソッド

※こいつは最初に必ず生成され，実行される.

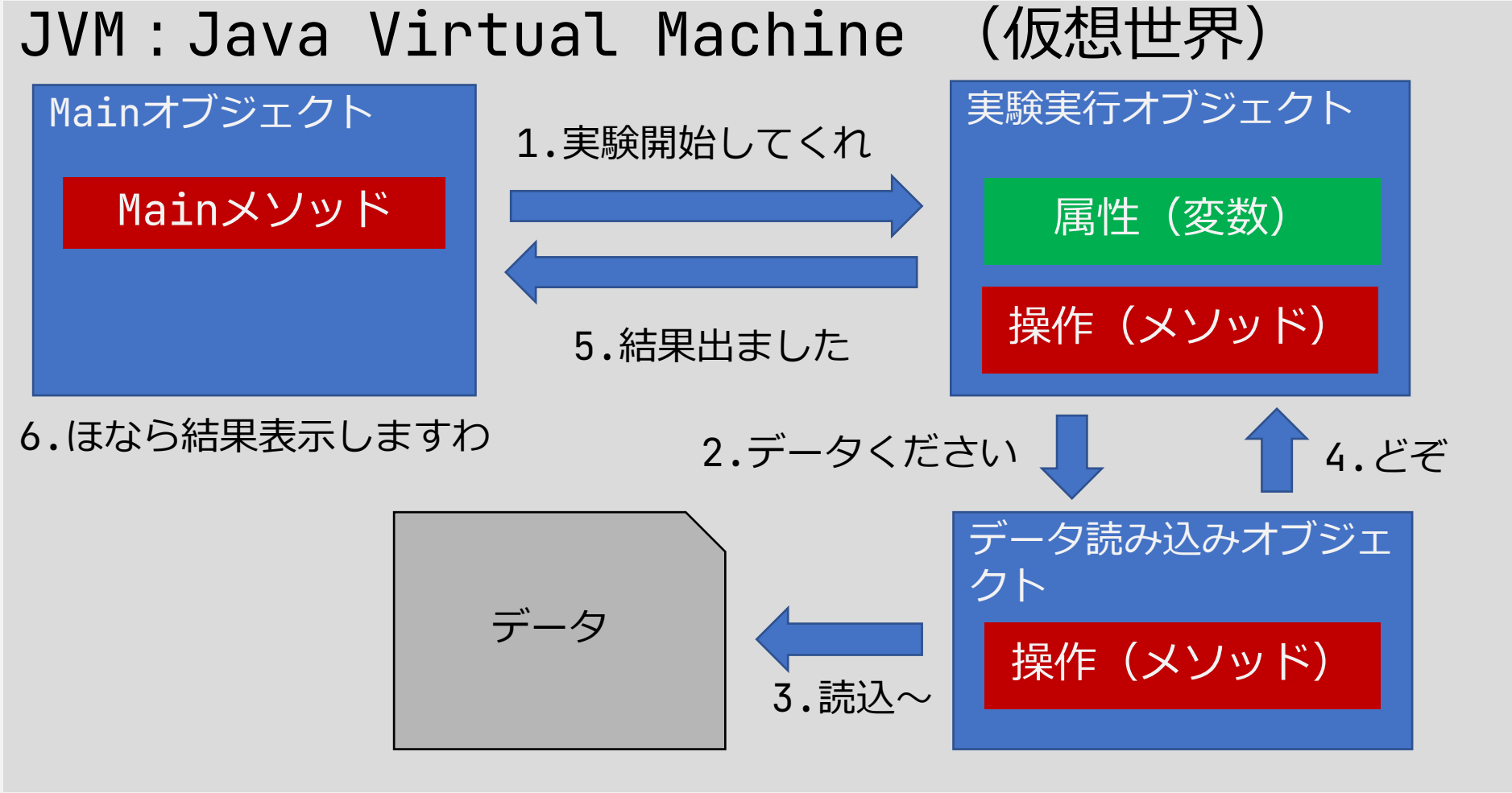
それぞれのクラス使ってそれぞれの役割持ったオブジェクトつくるよ～

実験実行オブジェクト
- データ読み込みオブジェクトにデータを要求
- 実験を実行

データ読み込みオブジェクト
- データを読込
- 要求があればデータを渡す

ざっくりとしたJavaの世界(2/2)

例えば実験でJavaを使うならこんな感じ.



オブジェクト指向って何がいいの？

オブジェクト指向はプログラムの変更を容易にし，プログラムの一部を簡単に再利用できる.

◆メリット１：柔軟性・保守性が高い

■過去に書いたプログラムを使って新たな機能に対応したプログラムを生み出せる.

■他人との認識共有がしやすい.

◆メリット２：再利用性が高い

■１からプログラムを書かずに，既存のプログラムを再利用できる.

⇒大規模なプログラムを安全に簡単に開発できる.

オブジェクト指向をなんとなく理解した

オブジェクト指向は他にもたくさんのメリットがあり，開発で大いに役に立っている.

オブジェクト指向にはこんな機能もある.

◆継承

■過去に作ったクラスを活用できる

◆多態性（ポリモフィズム）

■（継承があることで）似ているクラスを同じように捉え利用できる

◆カプセル化

■属性を他のクラスから操作できないようにする

- Pythonにはない？

後々詳しく取り上げる（かも）

第１章
悪しき構造の弊害

悪しき構造とは何か．また，それらが及ぼす影響について．

本章の目的

悪しき構造がどのような弊害を引き起こすかを知る．悪しき構造については次章から詳しく学ぶ．

◆弊害とは以下の様なものを指す．

■コードを読むのに時間がかかる

- 読み間違いも増えることになる

■バグを埋め込みやすくなる

- バグの温床になる

■悪しき構造がさらに悪しき構造を誘発する

- スパゲッティコード

本章の内容

内容とキーワード

- ◆意味不明な命名
  - ■技術駆動命名
  - ■連番命名
- ◆条件分岐のネスト
- ◆データクラス
  - ■低凝集
  - ■重複コード
  - ■修正漏れ
  - ■可読性低下
  - ■未初期化状態
  - ■不正値の混入

意味不明な命名（※詳しくは10章で）

わかりにくい命名はコードを理解する上で混乱を引き起こすのでやめよう.

◆以下の様な命名は意図が分かりにくくなる

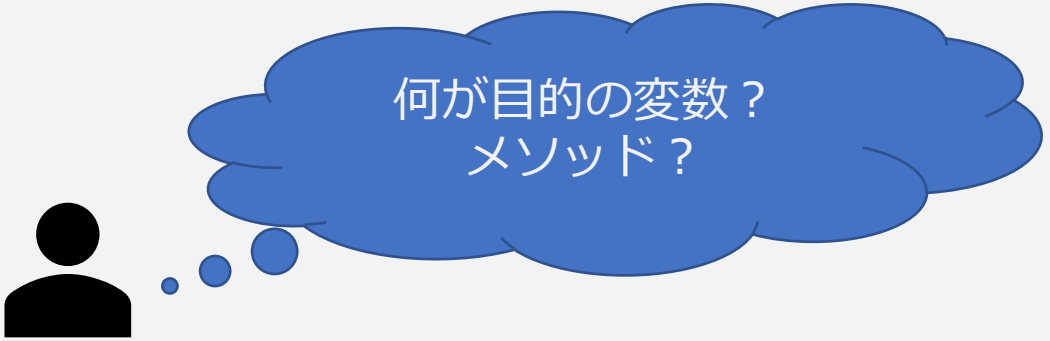
■技術駆動命名

- 型名を表すIntや，メモリ制御を表すMemoryやFlag等のプログラミングやコンピュータ用語に基づいた名前のこと.
- ※開発の分野によっては避けられない場合もある.

■連番命名

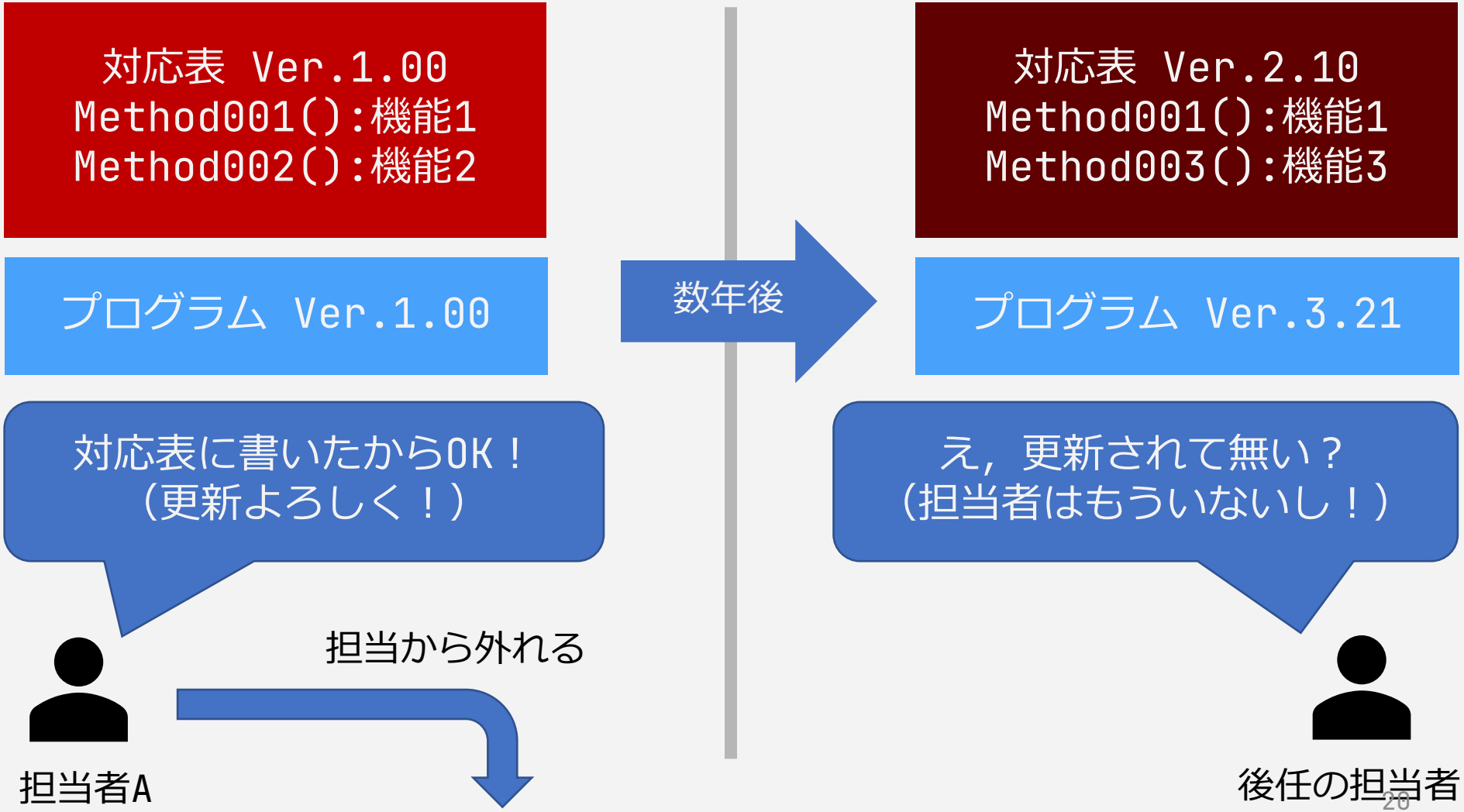
- Method1()のように番号付けで命名されているもの.

```
intValue01 = 0;
getState01Flag();
```



意味不明な命名と対応表

先ほどの意味不明な命名の対策として対応表を作成するという方法があるが，
対応表は更新が遅れがちで，対応表が嘘をつき始める．



条件分岐のネスト

ネストとは入れ子構造のことで，IF文の中にIF文を書いたりすることを指す．
ネストを多用するとコードの見通しが悪くなるのでやめよう．

◆実際にあるif文の多重ネスト

※プログラムの一部

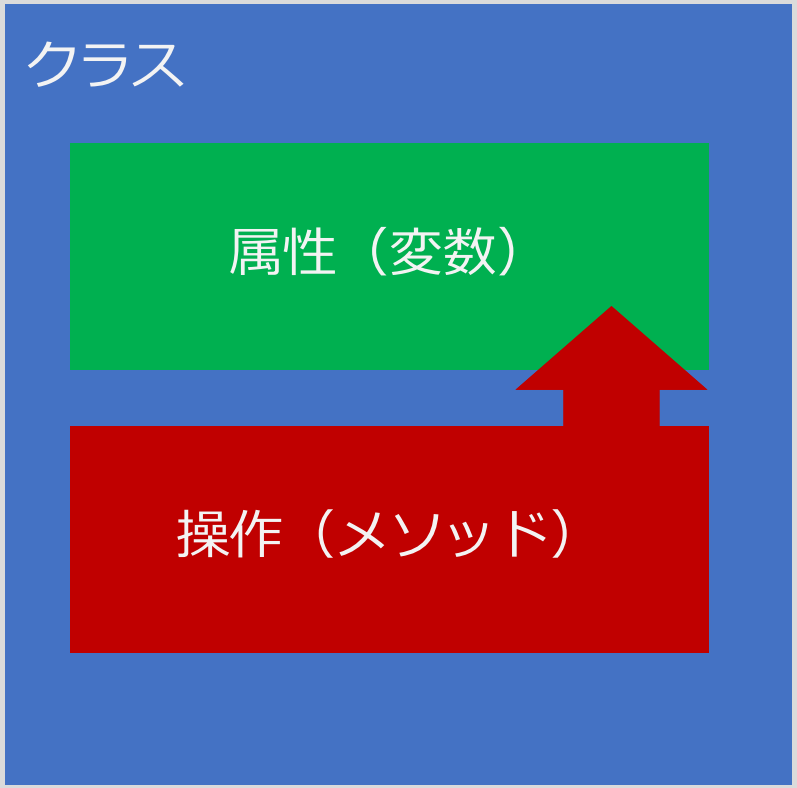
```
if(条件1) {
    何らかの処理（数十行）
    if(条件2) {
        何らかの処理（数十行）
        if(条件3) {
            何らかの処理（数十行）
        }
        何らかの処理（数十行）
    }
     何らかの処理（数十行）
}
```

条件1に該当する処理を変更したい人

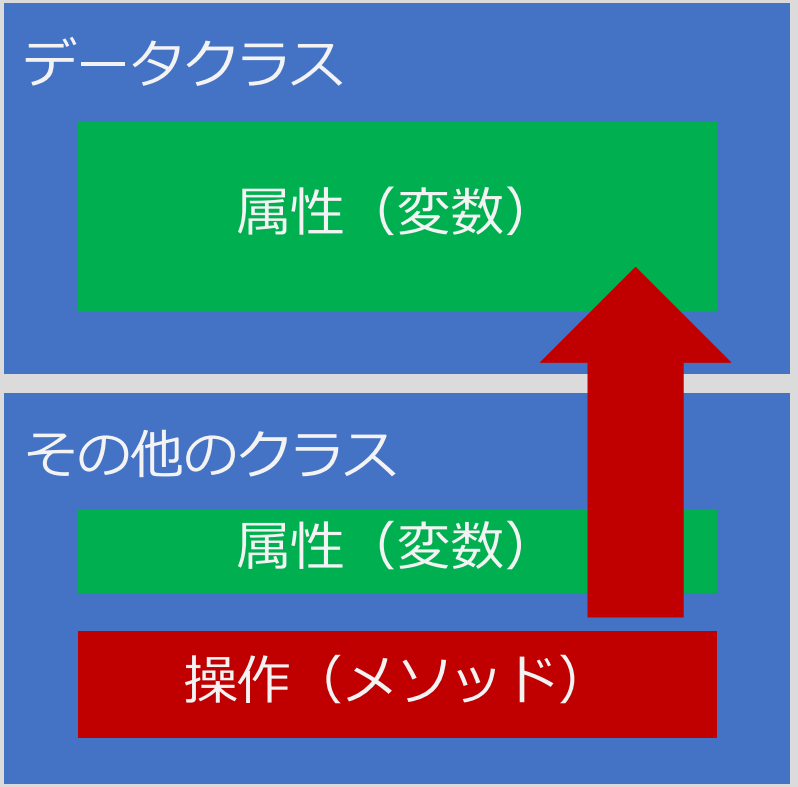
データクラス

データを保持するだけのクラスをデータクラスと呼ぶ．それらのデータを操作するメソッドを外部のクラスに作ってしまい，混乱を招く．（低凝集）

普通のクラス



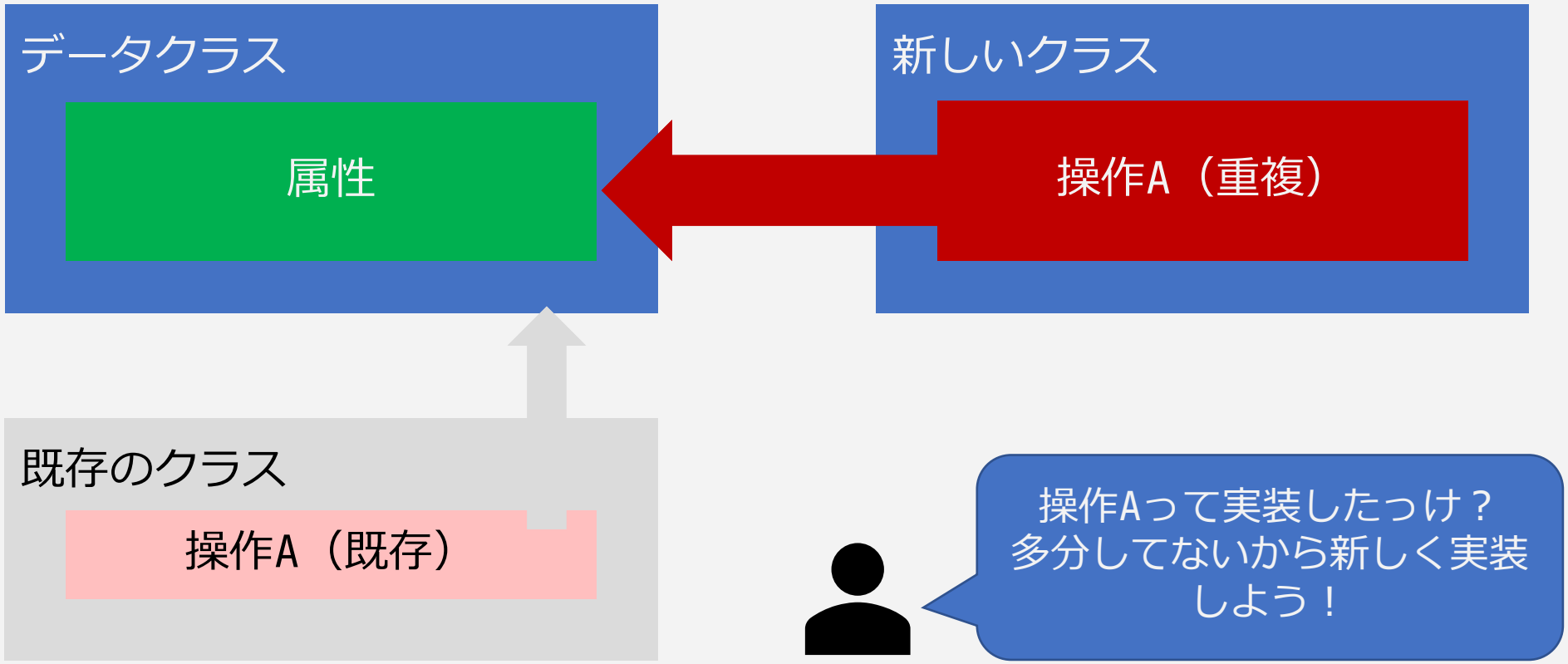
データクラス（低凝集）



低凝集が引き起こす弊害：重複コード

関連するコード同士が離れていると，重複コードが生じるようになる．

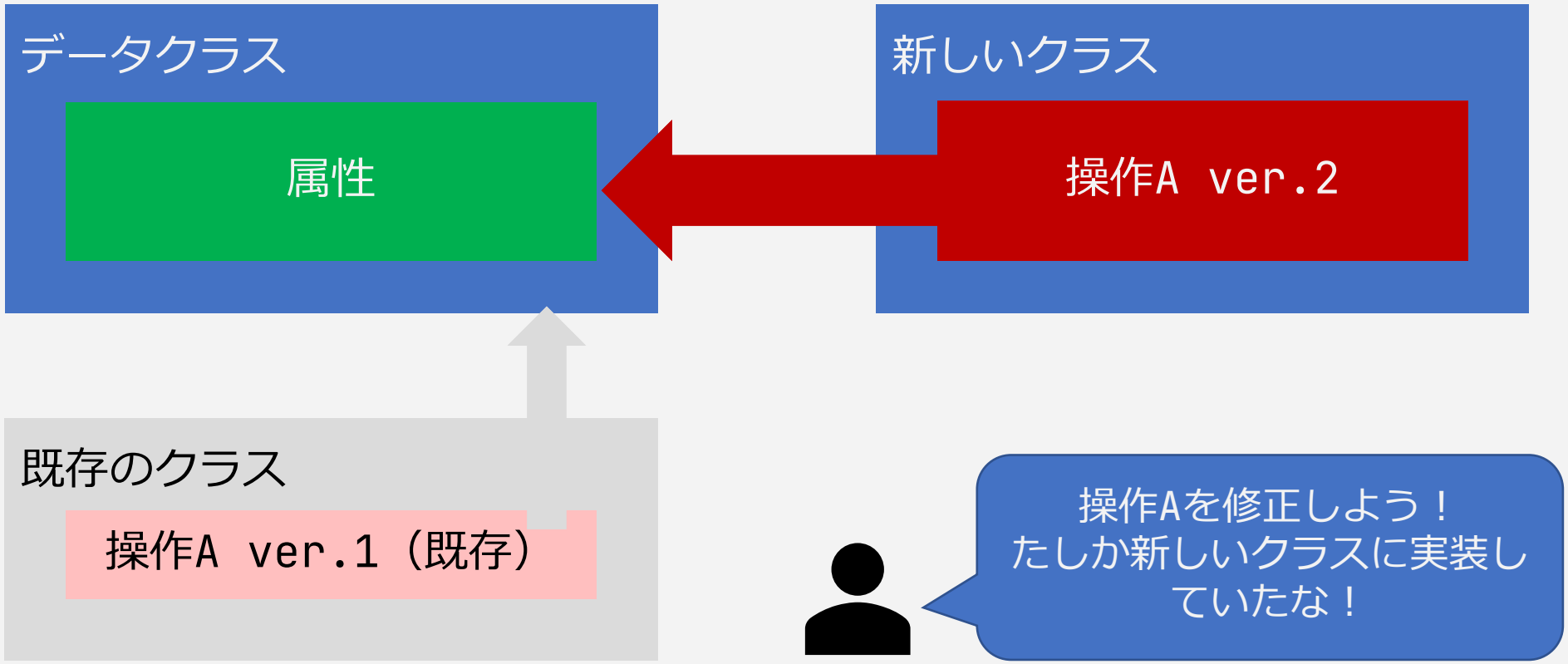
既に操作Aは実装されているのに…



低凝集が引き起こす弊害：修正漏れ

重複コードが多いと修正漏れが発生する.

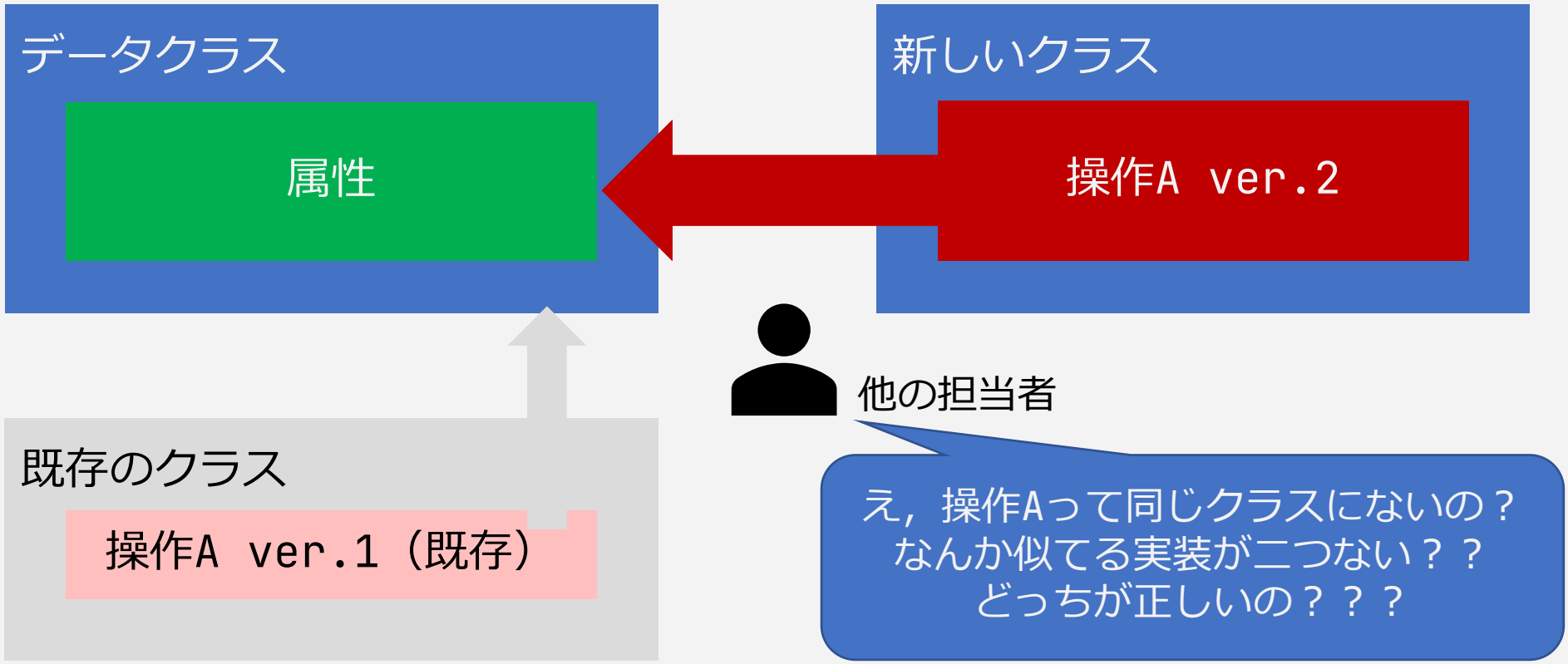
操作Aの内容を修正したいけど…



低凝集が引き起こす弊害：可読性低下

重複コードが多いと可読性が下がる．可読性とは，コードの意図や関係する処理の流れをどれだけ素早く正確に読み解けるかを表す指標．

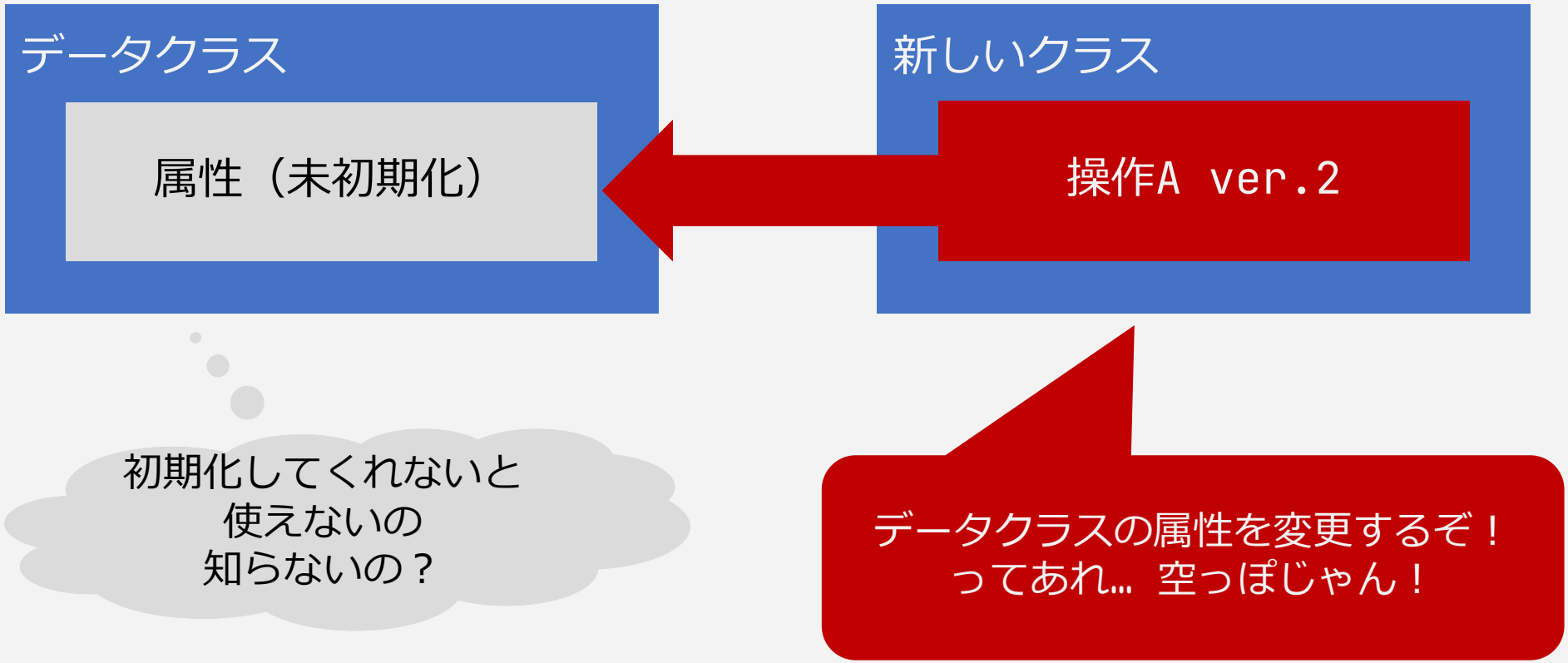
他の担当者が操作Aについて仕様を調査すると…



低凝集が引き起こす弊害：未初期化状態

クラスの属性（変数）は初期化しないとNull（空）のままなことがあり，そのままクラスを使用すると例外が発生する．（別名生焼けオブジェクト）

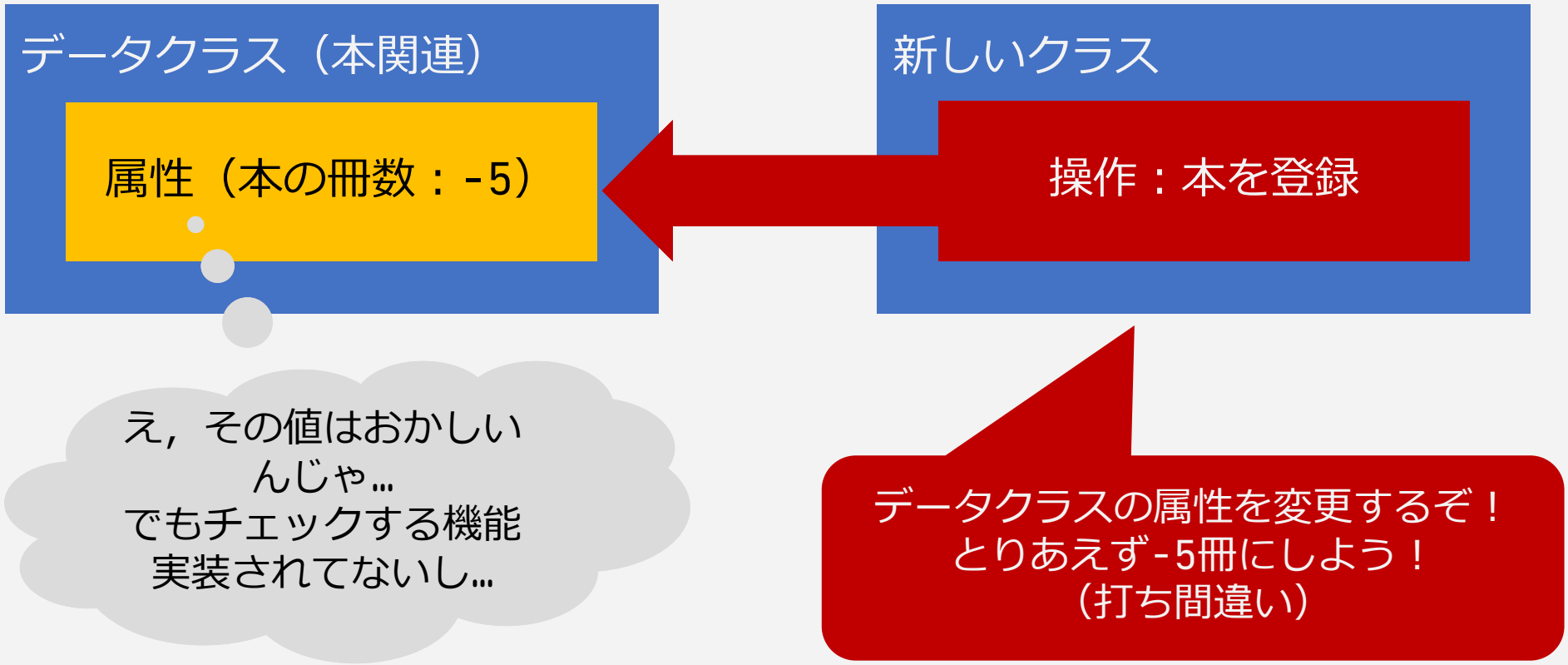
新しいクラスがデータクラスにアクセスしようとすると…



低凝集が引き起こす弊害：不正値の混入

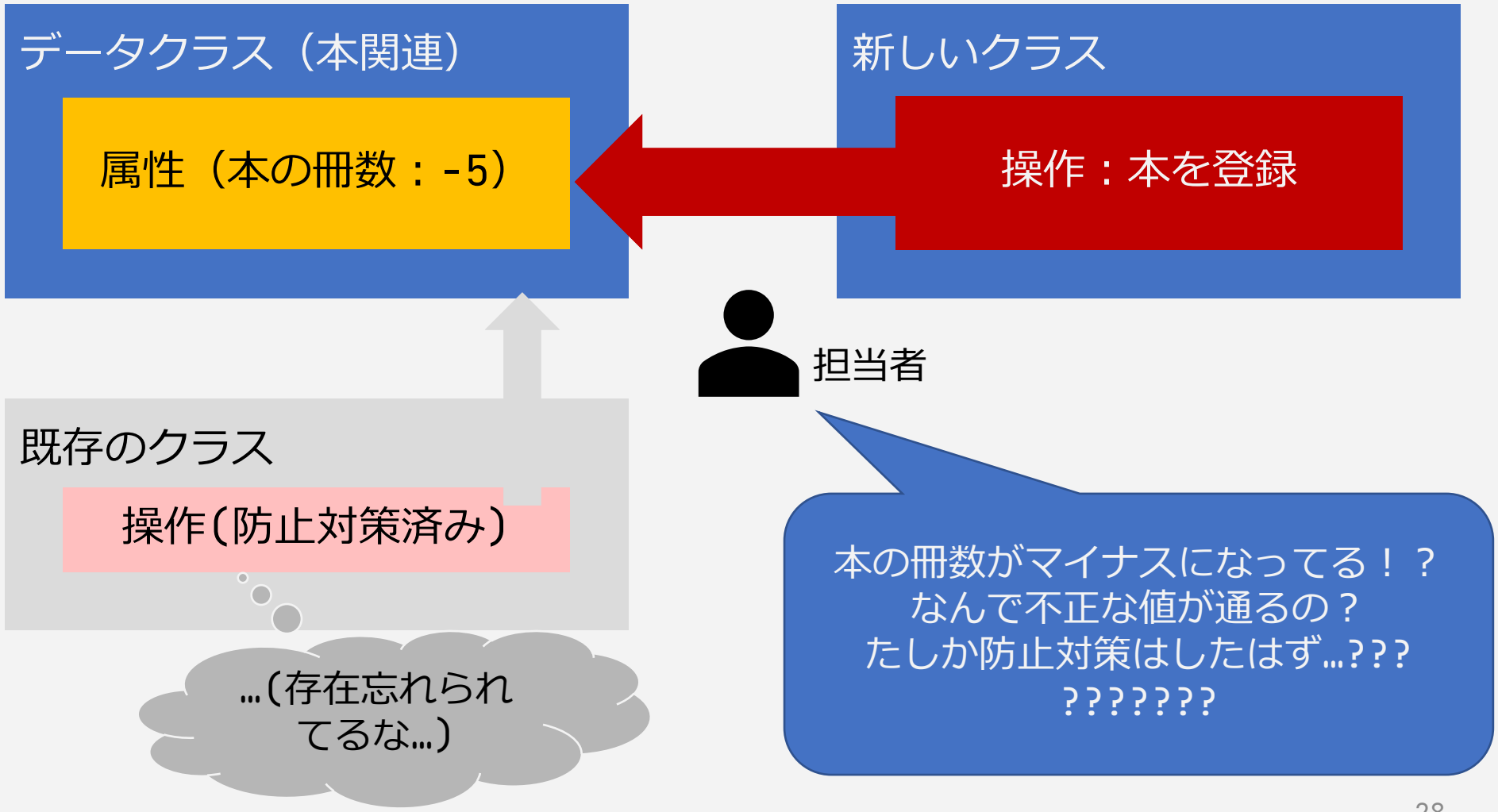
不正値の混入防止措置がとられていないと，不正な値を扱ってしまいバグを引き起こす.

例）新しいクラスが本に関するデータクラスに本を登録しようとすると…



低凝集が引き起こす弊害（最終的に...）

データクラスを作ってしまった末路.



次の章に向けて

悪しき構造が引き起こす弊害を簡単に紹介しました.

◆この章で取り上げたものは一部に過ぎない.

◆次の章から悪しき構造と設計方法について紹介.

◆悪しき構造の弊害を知ることで，それに対処するようになり，正しい設計方法を知ることで適切なコードを書けるようになる.