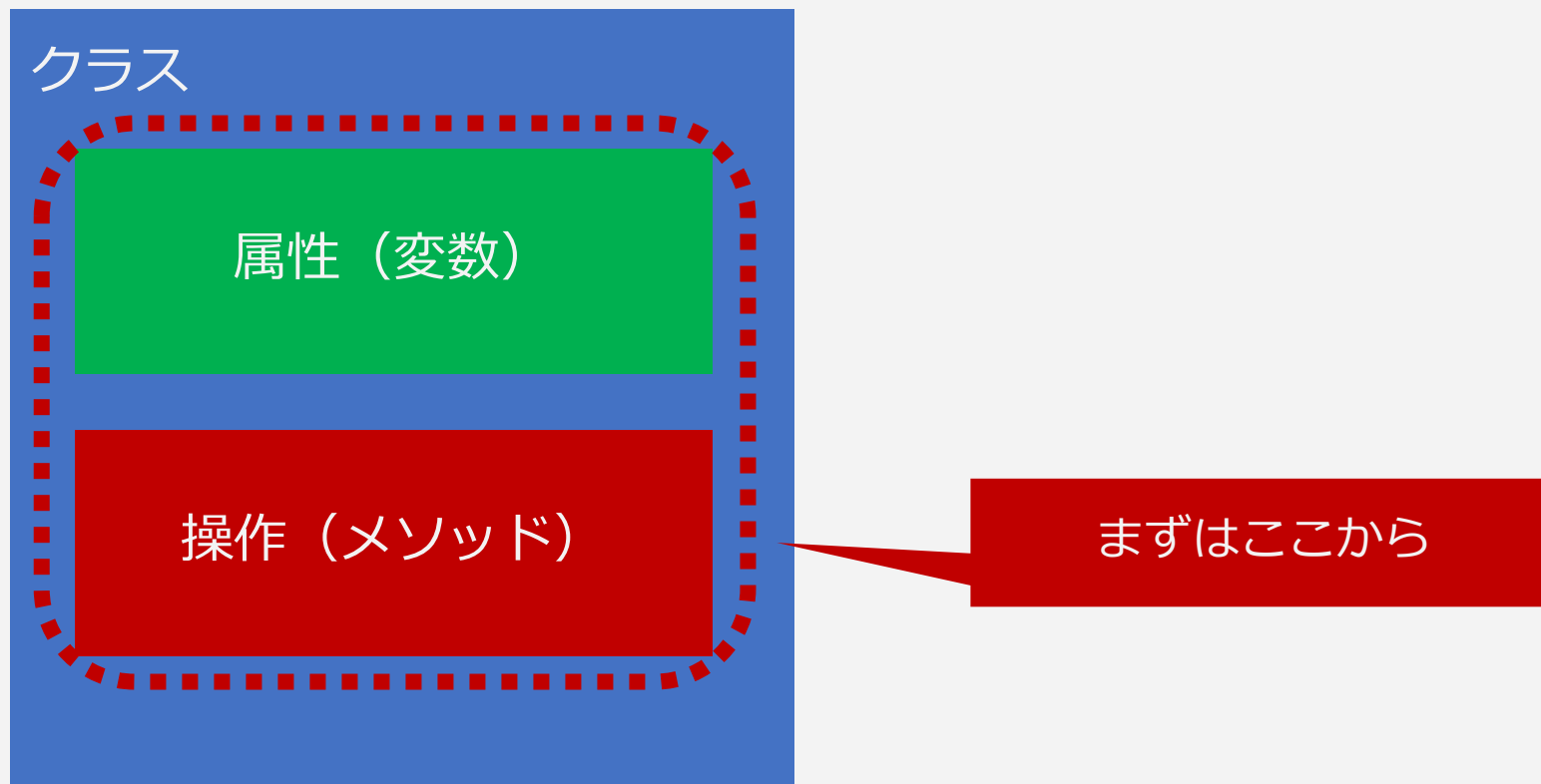


第2章 設計の初歩

設計の基本的な考え方

この章の目的

こういったことをするのが設計なのかを理解する。まずは変数やメソッド（関数）といった小さな単位的设计について学ぶ。



内容

内容とキーワード

- ◆意図が伝わる名前設計
- ◆目的ごとに変数を作成する
 - 再代入
- ◆意味のあるまとまりでメソッド化
- ◆高凝集なクラス

意図が伝わる名前設計（※詳しくは10章）

名前を省略しすぎると意図が伝わりにくくなるので、意図が分かる名前にする。（これも設計のひとつ）

省略しすぎると...

※プログラムの一部

```
int d = 0;  
d = p1 + p2;  
...
```

dってなんなんですか？
(あとp1,p2って何だよ...)

ダメージのDだよ！
(それくらい察してよ！)

意図が分かる名前にしよう

※プログラムの一部

```
int damageAmount = 0;  
damageAmount = playerArmPower  
+ playerWeponPower;
```

なるほどこれはダメージ量を計算
しているんですね...

そうそう！

目的ごとの変数を作成する (1/2)

変数に再び値を代入することを**再代入**（または**破壊的代入**）と呼ぶが、再代入すると変数の用途が変わるため、読み手が混乱してしまう。

再代入を多用すると...

```
※プログラムの一部
int damage = 0;
damage = power + weapon;
System.out.println(damage);
```

攻撃力

...


何らかの処理

...

実際のダメージ量

```
damage -= defense;
return damage;
```

※便宜上、変数名は簡単化しています



damageはpowerとweaponの合計
値なのかな？

読み進めると...



あれ、このdamageって何だっけ？
これ他にも再代入してる？

目的ごとの変数を作成する (2/2)

変数に再び値を代入することを**再代入**（または**破壊的代入**）と呼ぶが、再代入すると変数の用途が変わるため、読み手が混乱してしまう。

目的ごとの変数を用意しよう

※プログラムの一部

攻撃力

```
final int attack  
    = power + weapon;  
System.out.println(attack);
```

...

何らかの処理
(damageに再代入はない)

...

実際のダメージ量

```
damage = attack - defense;  
return damage;
```



attackはpowerとweaponの合計値か、これが素の攻撃力だね。



最終的にdamageがプレイヤーが受けるダメージ量なんだね。

変数の名前考えるのめんどくさい...

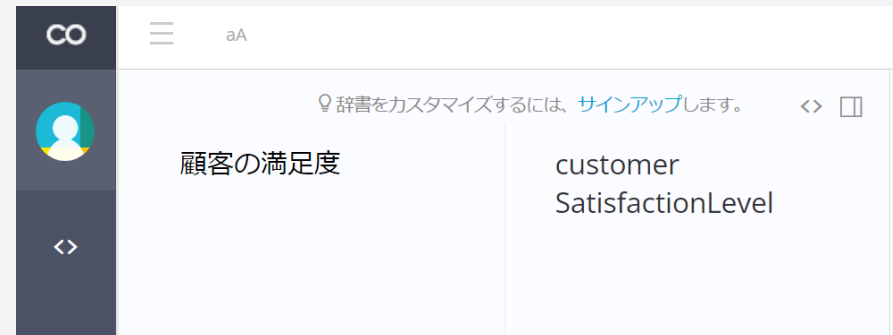
そういうときはツールを駆使しよう。

◆実は日本語から英語のネーミングを提案してくれるサイトがある。



codic-プログラマーのための
ネーミング辞書
(出典: <https://codic.jp/>)

使用例 :



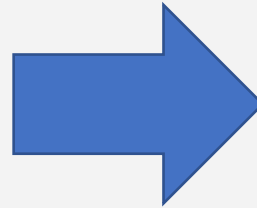
意味のあるまとまりでメソッド化

一連の処理の流れをそのまま書いてしまうと、どこからどこまでが何の処理が分からなくなってしまう。処理のまとまりをメソッドでまとめてみよう。

ごちゃごちゃメソッド

```
int Method() {  
  
    操作A (数十行)  
    操作B (数十行)  
    操作C (数十行)  
    return result;  
}
```

メソッドに
まとめる



整理整頓されたメソッド

```
int Method() {  
  
    操作A();  
    操作B();  
    result = 操作C();  
    return result;  
}
```

うーん結局なにやってんの？



なるほどこれは操作Aと操作B、最後に操作Cをしているのか...

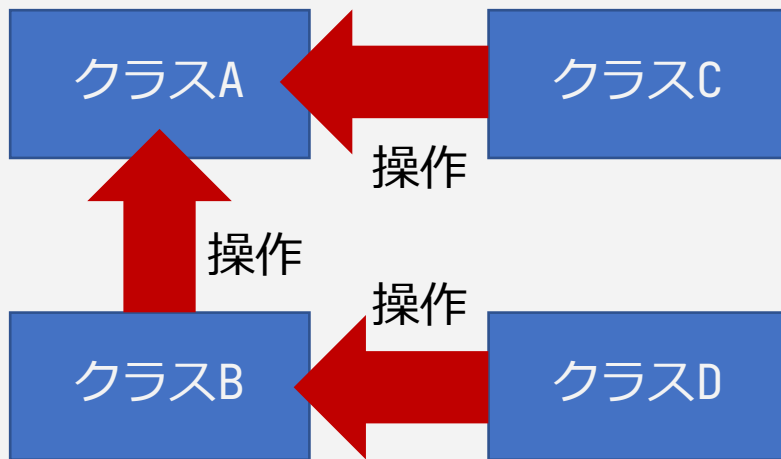


もちろんメソッド名も
明確なものである必要がある

高凝集なクラス

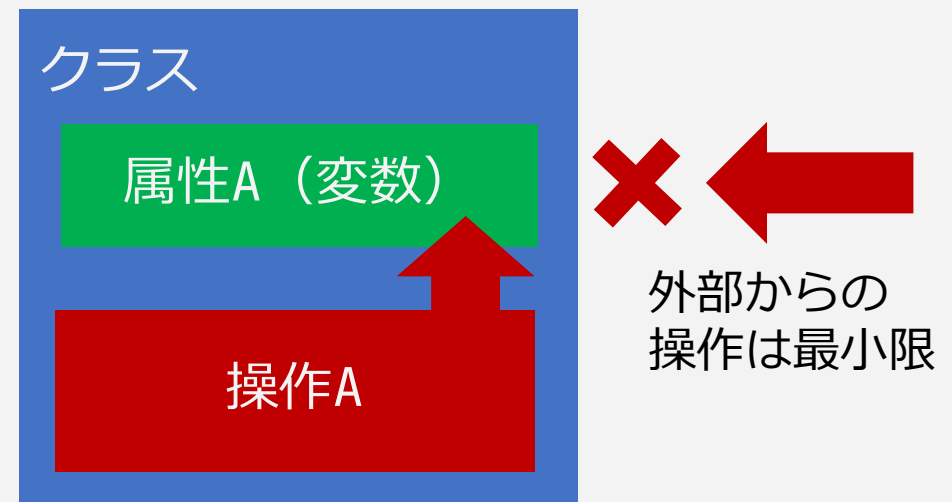
低凝集なクラスを作成してしまうとあちこちから影響を受けるため、重複コードなどのバグのもとが生まれてしまう。

低凝集だと...



クラスAの属性Aを変更する操作A
ってどこに実装したっけ...

高凝集ならば



クラスAの属性Aを変更する操作A
はもちろんクラスAにあるよね！

可読性が高い！

次の章に向けて

本章では設計の初歩を扱った。

- ◆名前を適当につけるのはやめよう
- ◆処理はまとめてメソッドに
- ◆低凝集にならないように気をつけよう
- ◆次の章ではクラスの設計方法を学ぶ