Lesson 1 クラスの復習

■お品書き

1. クラスの形
2. クラスの使い方
3. アクセス修飾子
4. コンストラクタ

[1] クラスの形

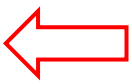
(1) クラス構文

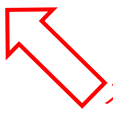
クラスは…

- データ(変数)を表すプロパティと処理(関数)を表すメソッドから成り立っている。
- メソッドの作り方は関数と同じ。
- プロパティとメソッドを合わせてメンバという。
- メンバにはpublicかprivateがつく(後述)。

クラスの構文は以下の通り。

```
class クラス名 {
    public/private データ型 $変数名;
            :
    public/private function 処理名(引数の型 $引数名, …): 戻り値の型 {
            処理;
        }
            :
}
```





(注)
関数内や実行部では、ソースコードの記述順に処理されていた。一方、クラス(インスタンス)内の記述順、関数記述ファイル内の関数の記述順と同じく自由。
可読性を考えると、プロパティをまとめてファイル上部に記述した方がいい。

(2) データ型

■ データ型には次のものがある。
- array: 配列
- bool: true/false(boolean)
- float: 小数値
- int: 整数値
- string: 文字列
- mixed: 型は何でも
- void: 戻り値なし

←疑似的な型

■ クラス型にはクラス名を記述する。

■ nullを代入する可能性がある場合は、前に?を記述する。

(ex)
- private ?string $name;
- public function showAns(?int $qNo): void {
- public function getName(): ?string {

■ 複数のデータ型を適用したい場合は、|で区切る(ユニオン型)。

(ex)

```
private int|string $numOrStr;    ←$numOrStrは整数か文字列
```

(注)
プロパティにデータ型を記述できるようになったのは、PHP 7.4以降。

(3) クラスとファイルのお約束ごと

クラス名は、大文字から始めるキャメル記法(アッパーキャメル記法)とする。

(ex)
- ○ class TestScore {
- × class testScore {
- × class test_score {

1ファイルに1クラスを記述し、ファイル名は、
クラス名.php
とする。

(注)
変数名(プロパティ名)、関数名(メソッド名)は小文字から始めるキャメル記法、定数は大文字のみのスネーク記法である。

[2] クラスの使い方

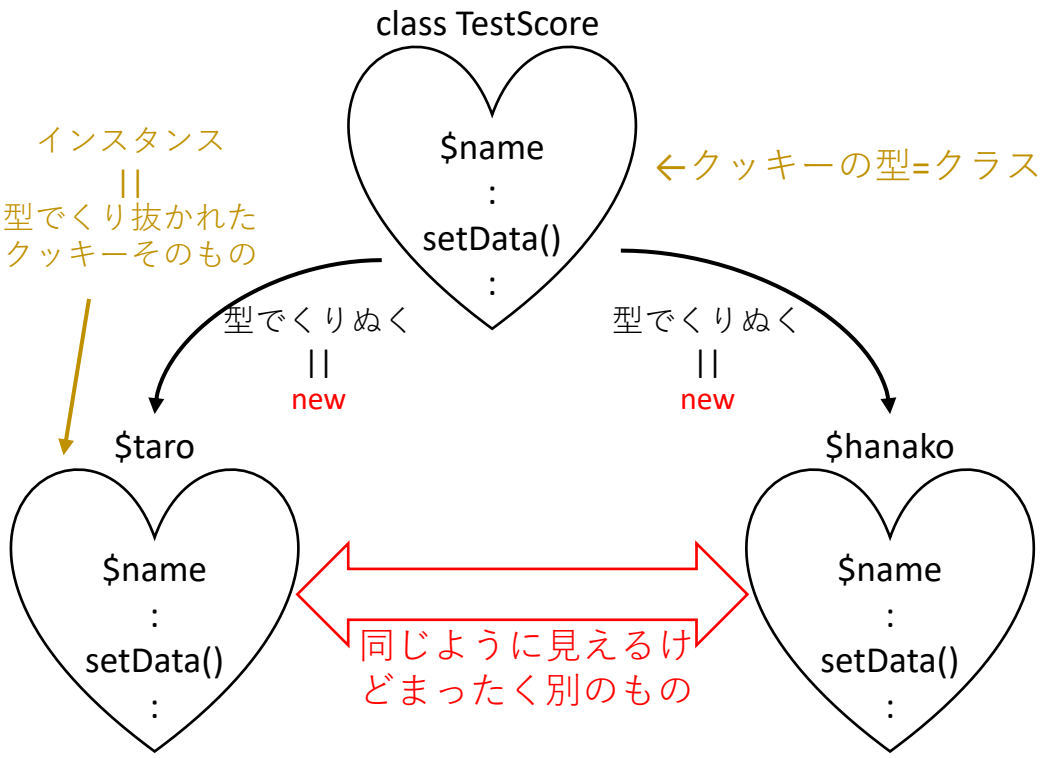
(1) クラスの利用はnew

クラスを利用するには、以下の通り。

- 以下の書式でクラスをnewしてそれを変数に格納する。
  new クラス名()
  クラスをnewしたものをインスタンスという(後述)。
  (ex)
  $taro = new TestScore();
- インスタンス内のメンバを利用するには「->」を記述する。
  (ex)
  - $taro->setData(…);
    →$taro内のsetData()メソッドを利用。
  - <?= $taro->name ?>
    →$taro内のプロパティ$nameを表示。

(2) クラスとインスタンスの関係

クラスとインスタンスの関係を図にすると…



オブジェクト指向はクッキーの型とクッキーの関係

- クッキーの型⇒クラス
- クッキーそのもの⇒インスタンス

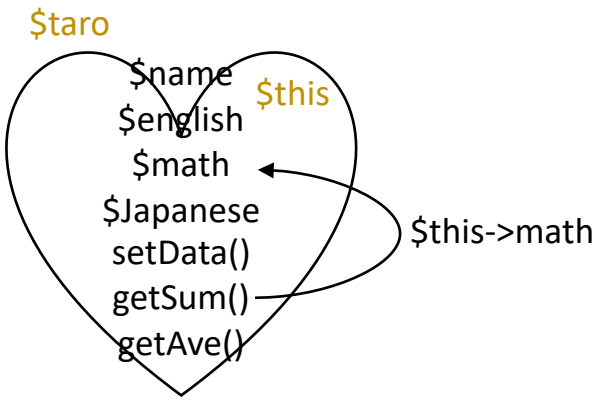
オブジェクト指向プログラミングでは、クッキーの型(クラス)を作り、その型(クラス)からクッキー(インスタンス)を作り出して使っていく。
また、既存の型(クラス)を使うこともよくある。

(3) インスタンス内のアクセスはthis

インスタンスの外部からそのメンバにアクセスする場合は、
　そのインスタンスが格納された変数->メンバ名
となる。
一方、インスタンス内部からのアクセスは、
　$this->メンバ名
となる。このthisは自分自身を指す。

(ex)
- return $this->english + …;
- $ave = $this->getSum() / 3;

こんなイメージ…



[3] アクセス修飾子

(1) publicとprivate

Src3(showTestScore.php)処理部最終行に以下の行を追加したら、ありえない結果を表示してしまった。

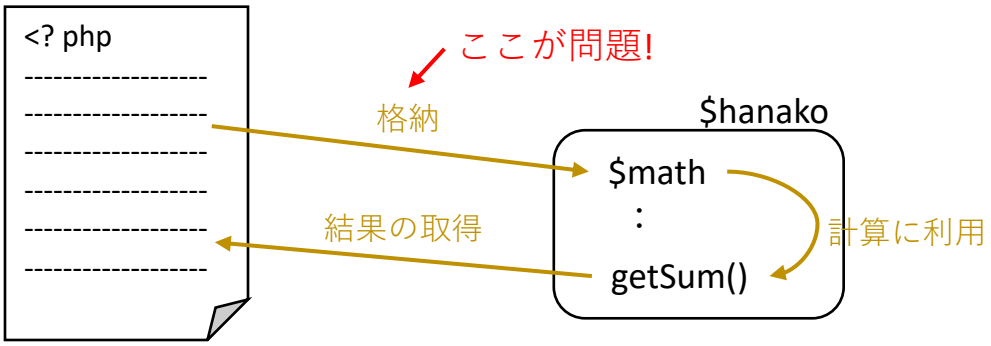
```
$hanako->math = -35;
```

さらに、以下の行を記述したら、エラーとなった。

```
$hanako->math = "いえい";
```

なぜか?

プロパティ$mathはそもそも整数値を想定している。



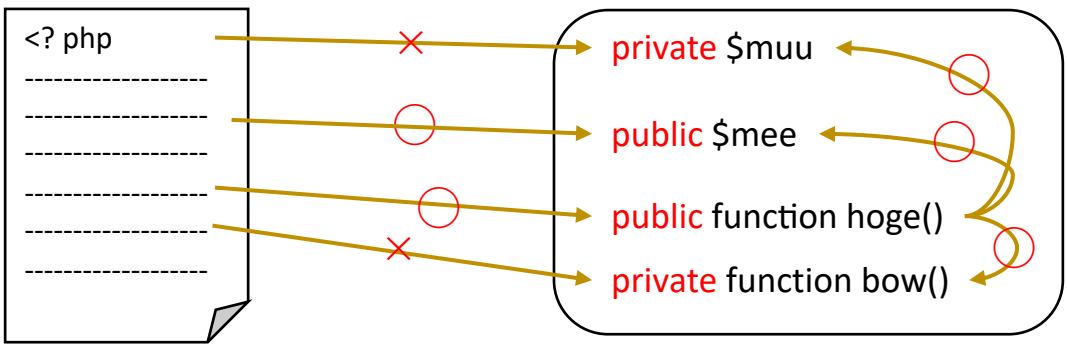
プロパティに外部から直接アクセスできる状態だと、不適切な値が格納されてしまう。これを防ぐには…

クラスメンバに対して外部からアクセスできるかどうかを制御するのがアクセス修飾子であり、メンバの前に以下のキーワードをつける。

- public→クラス内外を問わずどこからでもアクセスできる
- private→クラス内からのみアクセスできる

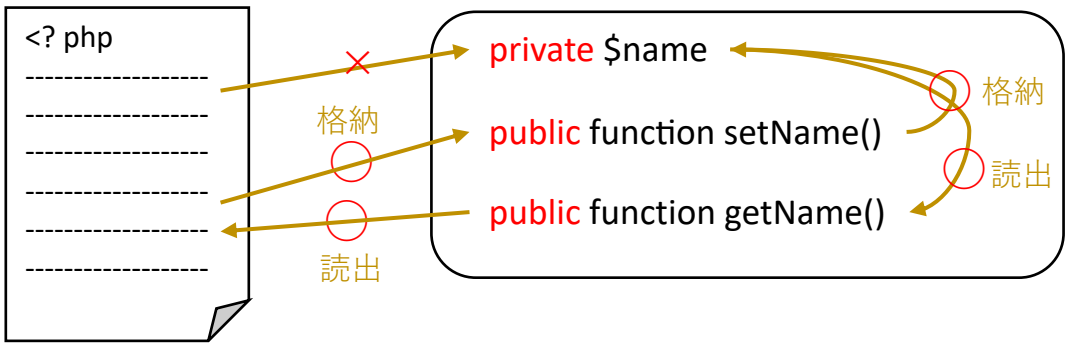
他に、protectedというのもある。

こんなイメージ



(2) アクセサメソッド

Src5(TestScoreEncapsulationクラス)はプロパティをすべてprivateにして外部からアクセスできないようにしている。では、プロパティの値の受け渡しはどうするか?
これは、それ用のメソッドを用意する。



不適切なデータを格納されないために、プロパティをprivateにし、代わりに、プロパティにアクセスするためのアクセサメソッドを用意する。これを、カプセル化といい、オブジェクト指向言語の三大特徴のひとつ。

アクセサメソッドの作り方は以下の通り。

- データを格納するためのメソッドをセッタといい、メソッド名は
  setプロパティ名()
  とする。引数はプロパティと同じものとし、処理内容は引数をプロパティに代入する。
  (ex)
  ```
  public function setName(string $name): void {
      $this->name = $name;
  }
  ```
- データを読み出すためのメソッドをゲッタといい、メソッド名は
  getプロパティ名()
  とする。引数はなしで、処理内容はプロパティをリターンする。
  (ex)
  ```
  public function getName(): string {
      return $this->name;
  }
  ```

アクセサメソッドに型指定をしておくと、不正なデータをセットしようとするとエラーで教えてくれる。
そのため、バグ発見が早くなる!

[4] コンストラクタ

TestScoreEncapsulationクラスには欠陥がある。
それは、newした後にsetData()メソッドを実行し忘れると、プロパティにデータが格納されずに、必要な計算ができない。この場合、エラーにすらならない。
必要なデータを確実にもらえるタイミングとはいつか?→newの時!

new時(インスタンス生成時)に何か処理を行いたい場合は、コンストラクタという特殊なメソッドに処理を記述する。
このコンストラクタに引数を設定すると、newの時に受け渡しが可能となる。

PHPのコンストラクタは以下の形。

```
public function __construct(引数の型 $引数名, …) {
    newの時に行いたい処理
}
```

特殊なメソッドなので、戻り値はない。つまり、returnは記述しない。

(ex)

```
public function __construct(string $name, int $english, …) {
    $this->name = $name;
    $this->english = $english;
        :
}
      ⇓
$taro = new TestScoreConstructor("たろう", …);
```

newの時にこのコードが実行される

引数が設定されているので、new時に渡す必要がある

コンストラクタで引数が設定されている場合、new時に引数を渡さないとエラーとなる。
×$jiro = new TestScoreConstructor();

なし