クラスの解説ハイライトと教え方のコツ

2023年04月17日

PHP技術者認定機構
理事 CTO 古庄 道明

始めに

- プログラミングの学習で「ここを教えるのが難しい」「ここで躓きやすい」という箇所は、ぶっちゃけ沢山あります
- 「クラス」はそんな風に「躓きやすい」箇所なのに加えて、教えるべき事柄も教えたい事柄も沢山あるので、なおのこと「学ぶ人」と「教える人」とのギャップが出やすいのではないか、と思います
- 今日はその辺りのギャップを多少なり埋められるようなお話ができれば、と思います

- 今回のお話は「教える人」の側の目線が多いため、「クラスを学習する人」には少しハイペースになるのでご了承ください

自己紹介とプロフィール

- 古庄道明(ふるしょう みちあき)
  - ネット上だと、がる(gallu)というハンドルを使う事が多いです
- コンピュータに触ったのは小学生の頃で、その時にN88 BASICなど(正確にはN6x BASIC経由N88 BASIC)で写経したのが始めてのプログラミングです
- 学校卒業後は、営業や接客などのお仕事をしていました
- そこから紆余曲折あり1995年からエンジニアをやってます
- わりと昔から「現場で教育とレクチャー」はやってました
- 2012年からご縁があり専門学校の講師を兼任しています
  - 1年ほどですが、職業訓練校の講師もしていました

クラス? オブジェクト指向?

- よく使われる「クラス」と「オブジェクト指向(プログラミング)」という単語ですが、実は「オブジェクト指向」という単語自体は、PHP公式の言語リファレンスには出てきません
  - 関数リファレンスには「オブジェクト指向型」「オブジェクト志向のインターフェイス」といった表現が出てきます
- なので、教える時は出来るだけ「クラス」に呼び方を統一しておくとよいでしょう
- ただ外界では「オブジェクト指向(プログラミング): OOP」という言い方もよく見かけるので「単語は見覚えておけ」くらいは伝えてもよいでしょう
  - 「厳密に」この辺(OOP)のお話をすると割と深くて濃い内容になるので、いったんはあまり踏み込みません(笑

「実装」と「設計」を混ぜない

- クラスを実際に実務で用いる時に、問題になったり議論になったり、そのために学んだりするのは大体「設計」にまつわる事柄です
- しかし、初学者がまず学ばなければならないのは「実装のための文法」です
- なので、教える時はまず「実装」に絞って教えるように、十二分に心を砕きましょう
- 「ある程度の実装」を教えてから「導入部分くらいの設計」を教えて、それが学び取れたら「高度な実装」を学んで「高度な設計」を学んで……という風にサンドイッチしていくとよいと思います

「学習の前提」は確認をしておきましょう

- ものすごく暴論ですが、クラスは概ね「定数」「変数」「関数」の集合体(+α)です
- そのため「定数」「変数」「関数」がわかっていないと理解がものすごく大変です
- なので、まず前提として「定数」「変数」「関数」を(ある程度)理解していることを確認しておきましょう
  - 関数の教え方は直前の三雲さんのセッションを再度ご確認ください(笑

とはいえ「闇雲に」学んでも頭に入らない……

- 「何に使うか」がわからないとなかなか学習に身が入らず、脳に残らないものです
- ただ一方で「ある程度知識がないと」、こんな時に使う……というお話をしても理解できないのもまた事実です
  - クラスに限らない、学習全般に言えるお話ではあるのですが……
- 教える時は大体、次の2パターンのどちらか(か、その亜種)を選択する事になるかと思います

「まず基礎を学べ」パターン

- 一般的に「教える」時によく使われるパターンです
- 「まず文法をしっかり教えて」「その後に使い方を教える」パターンです
- 「教える側にとっては説明しやすく楽」な反面「学ぶ側にとっては砂をかむような時間が長いのでシンドイ」のが難点です

- 改良パターンとして以下があります
  - まず文法を「後でもう一度やるから」とざっと教える
  - 使い方を教える
  - その「使い方」に必要な文法を「遡って」もう一度教える
- よく使う文法はよく遡るので、学習効率は上がります

サンドイッチパターン

- 「文法」と「使い方」を細かく分解して、交互に教えていくパターンです
- うまくはまるとモチベーションの維持がしやすいのですが……
- 特に初級の頃は「使い方が不自然だったりわざとらしかったりあまり有用とは思えなかったりする」感じの使い方になってしまう事も多いので
- そのあたりの設計力と分解力と経験と想像力が要求されます

ハイブリッドパターン

- 個人的には「その中間」くらいが楽かと思っています
- 「まず基礎」パターンを貫くとこぼれ落ちやすいのですが
- サンドイッチパターンにするには、初期のクラスの学習においては「有効な実例」がなかなか作りにくいので

- 「ある程度サンドイッチ」にしつつ1つの「文法->使い方」の中では「まず基礎(の改良パターン)」を使って「文法ざっと->使い方->文法の復習」と学習をしていきます
- これだと「さほど不自然な使い方をひねり出さなくてもすむ」ので、教えるほうの負担も減るかと思います

ハイブリッドパターンの一案

- きっと色々な先生や教師講師ごとに色々な切り口があると思います
- その一例として「例えばこんな切り口」というのを紹介していきます

- 少しの間「教える」モードで記述していきます
  - 「教える側向けの発言」は、括弧書きの中に書いていきます

空っぽのクラスをつくる

- クラスはまず大前提として「こんなクラスを作るよ」という宣言をします
- そして、その宣言に「new」という命令(キーワード、と呼称します)を使って、インスタンスまたはオブジェクトというものを生成していきます
  - (マニュアルでも「インスタンス」と「オブジェクト」はどちらも出てくる用語なので、「基本PHPでは同じ」事を説明しましょう)

```
<?php
// クラスを定義する
class クラス名 {
}

// インスタンスを生成する
$インスタンス = new クラス名();

// 生成したインスタンスを確認する
var_dump($インスタンス);
```

クラスは変数を持つ事ができる

- クラスの中には変数が(複数)定義できます
  - 「プロパティ」という言い方があるので覚えておいてもよいでしょう
- (いったん「アクセス権」はpublicで進めます)
- 他の言語をやっている場合(特にC言語など)、「構造体」という言い方をするとわかる人がいる、かもしれません
- インスタンスの中にあるプロパティに対して、データを入れたり出したりしてみましょう
- (型宣言は、関数の時にすでに学習していたら取り入れる、まだ未学習なら取り入れない、でよいと思います)
- コードは次のページで

```php
<?php
// クラスを定義する
class クラス名 {
    // プロパティを定義する
    public $hoge;
    public $foo;
}

// インスタンスを生成する
$インスタンス = new クラス名();
// 生成したインスタンスを確認する
var_dump($インスタンス);

// プロパティに値を代入する
$インスタンス->hoge = 'string';
$インスタンス->foo = 999;
// プロパティの値を確認する
var_dump($インスタンス->hoge);
var_dump($インスタンス->foo);

// インスタンス自体を確認する
var_dump($インスタンス);
```

どんな時に使うの?

- 「ひとかたまりのデータ」を、そのまま「かたまり」としてあつかう時に便利です
- (例えばBBS等の「1つの発言」やECサイトの「1つの商品」「1生徒の期末テストの結果」「1キャラクターのパラメータ(ゲーム)」などの、学ぶ側が理解しやすいサンプルを用意するとよいでしょう)
- 例として「BBSの発言」でみてみましょう
  - 発言日、発言者名、タイトル、発言内容、ユーザエージェント、通信元IPアドレス
- コード例は次のページへ

```php
<?php
// クラス定義
class BBSの発言 {
    // プロパティを定義する
    public $発言日;
    public $発言者名;
    public $タイトル;
    public $発言内容;
    public $ユーザエージェント;
    public $通信元IPアドレス;
}

// クラスを使った時の処理の例
$BBSインスタンス = new BBSの発言();
// インスタンスにデータを入れたりする処理
XXX
// 例えば「(どこかに)書き込みをする」
書き込む関数($BBSインスタンス);

// クラスを使わなかった時の処理の例
// データを取得する処理
XXX
// 例えば「(どこかに)書き込みをする」
書き込む関数($発言日, $発言者名, $タイトル, $発言内容, $ユーザエージェント, $通信元IPアドレス);
```

- まだ引数が1個 vs 6個だから「ん……どっちでも……」と思うかもしれません
  - でも、処理って「書き込む」以外にも色々あるんですよね
- でもこれがもし、20個だったり50個だったり100個だったりしたら?
  - 個人的には、不動産系で「150個以上の要素がある」という経験があります
- さらにここで「要素が増える」と、修正の手間がずいぶんと変わります
  - クラスなら基本「クラス定義を変える」である程度対応できますが、関数だとまず「全部の関数の引数を修正する」から始める必要があります

- こんな風に、クラスは「データをまとめてひとかたまりにして」扱えるのが、まず便利です
- また「ひとまとめ」なので、修正する時も「どこを修正すべきか」がわかりやすいので、変更があった時に対応しやすいのも大きな利点です
- (という風に、メリットを説明すると刺さりやすいでしょう)

「カプセル化」についての説明をどうするか?

- (ここだけ全体的に「教える側向け」です)
- この辺りで悩むのが「カプセル化」の説明についてです
  - ざっくり書くと「詳細の隠蔽」
- 後でその辺は「設計」の時などでお話をするタイミングがあるので、この時点では特に説明しないことが多いです

クラスは「処理(関数)」を持つ事ができる

- クラスの中には関数が(複数)定義できます
  - 「メソッド」という言い方があるので覚えておいてもよいでしょう
- 書き方は「関数」とほぼ一緒です
- メソッドを実装して、メソッドを呼び出してみましょう
- コードは次のページで

```php
<?php
// クラスを定義する
class クラス名 {
    // メソッドを定義する
    public function hogeFunc() {
        echo "call hogeFunc <br>¥n";
    }

    public function fooFunc($num) {
        return $num ** 2;
    }
}

// インスタンスを生成する
$インスタンス = new クラス名();
// メソッドを呼び出してみる
$インスタンス->hogeFunc();
// メソッドを呼び出して戻り値を受け取る
$ret = $インスタンス->fooFunc(10);
var_dump($ret);
```

「自分のデータに対する処理」を持つ事ができる

- 今までは「変数(プロパティ)」と「関数(メソッド)」をそれぞれ別々に実装しました
  - (どこかで置き換えていくのですが、しばらくは「変数/関数」と「プロパティ/メソッド」という言い方は併用しておくと、置いてきぼりになる人が減ると思います)
- 次は「自身の変数(プロパティ)を使った関数(メソッド)」を実装してみましょう
  - $this という新しい書き方が出てくるので覚えましょう
- コードは次のページで

```php
<?php
// クラスを定義する
class Person {
    // プロパティを定義する
    public $last_name;
    public $first_name;

    // メソッドを定義する
    public function getName() {
        // 自身の中にある last_name と first_name を結合してreturnする
        return $this->last_name . ' ' . $this->first_name;
    }

}
// インスタンスを生成する
$person_obj = new Person();
// プロパティにデータを設定する
$person_obj->first_name = '道明';
$person_obj->last_name = '古庄';
// メソッドを呼び出す
$name = $person_obj->getName();
var_dump($name);
```

どんな時に使うの?

- 中のデータを「こんな風に加工したい」という状況は、とてもよく出てきます
  - 税別価格の「税込み価格計算」をしたい
  - 開始時間と終了時間から「所要時間」を計算したい
  - 投稿した文に「禁止ワード」が入ってないか調べたい
  - (上述の1番目と2番目は、実際に実装をして学習させてみるのもよいと思います)
- そのため、この「プロパティを定義、メソッドで〝自身のプロパティ〝も使った処理を書く」というのは、非常によく使われます

- モードを「教える側向け」に戻していきます

- 業務で「プロパティがpublic」というのは極めて珍しいと思うので、この辺りで「private」を教えます
- ただそうすると「データの代入」が出来なくなるので、以下の順番で知識を増やしてもらう事が多いです
  - まずは「セッター」「ゲッター」のメソッドを作成して、メソッド経由でのプロパティのアクセスができるようにします
  - コンストラクタを「概念」「書き方」とも学習してもらい、「コンストラクタのタイミングでデータを入れる」事を覚えてもらいます
    - そのタイミングで「セッターをどうするか」というのがありますが、最近の風潮的には「1プロパティへのセッターを作る、が比較的、忌避されやすい傾向がある」ように思うので、コンストラクタ作成のタイミングでセッターは削ってもよいかと思います

例外は厳密には「クラス」とは別ものですが

- コンストラクタでエラーが起きた時は「例外を投げる」、を含めて、例外が出てくるケースはこれから増えてくるので
- 大体このあたりのタイミングで「例外の概念、投げ方とcatchの仕方」をやります
  - いったんは「最低限の書式」がわかるくらいでよいかと思います

- 最後に「定数」の書き方を学習してもらうと、いったん「本当に最低限」程度の「クラスの文法」は一区切り学べるのではないか、と思います

- 続いて「継承とインタフェース」とかの学習に進むのですが
- ここまでの学習が「ある程度しっかり」身についていないと厳しいので、先に進む前に「色々な(簡単な)クラスを書いてみる」といった事を何度かやって、「簡単なプロパティ+メソッド+定数のクラス」ならある程度スムーズに書ける、くらいまで習熟しておいてもらうとよいでしょう
  - 「まずは量を書く」というのはいつでも大切です

継承について

- 継承は、実装から入っても使い方から入ってもよいのですが
- 今回は「使い方->実装」の順番で学びのルートを確認してみましょう

is-a関係について

- 継承にとって「is-a関係」はとても重要な考え方なので、まずは「is-a関係」についてレクチャーしましょう
- 「A is a B」の関係です……で終わるとイメージしにくいので、例を色々と挙げていくとよいでしょう
  - A is a B(の一種)
  - 白金豚バラ肉 は 豚肉(の一種) です
  - 豚肉 は 食肉(の一種) です
  - 奥久慈しゃも は 鶏肉(の一種)です
  - 鶏肉 は食肉(の一種) です
  - 食肉 は 食べ物(の一種) です
- この「複数のAに共通するBを見つけ出す」事を「汎化(はんか)」と言うので、この辺も説明しておくとよいでしょう

継承の使い方の例

- よく「ECサイトの商品」で説明をしますので、例題として
- 書籍を扱っています。書籍には以下のデータがあります
  - 書名、著者名、出版社、発売日、ISBN、書影、価格
- 家電をあつかっています。家電には以下のデータがあります
  - 商品名、メーカ、発売日、商品コード、消費電力、外見画像、価格

- まずは「それぞれ別々に」必要なプロパティを持つクラスを作成します

- 書籍
  - 書名、著者名、出版社、発売日、ISBN、書影、価格
- 家電
  - 商品名、メーカ、発売日、商品コード、消費電力、外見画像、価格

- 「著者名」は書籍に、「消費電力」は家電に固有のデータ
- 「書名と商品名」「出版社とメーカ」「ISBNと商品コード」「書影と外見画像」は、言い方は違うが同じ意味のデータ
- 「発売日」「価格」は同じデータ
- 上述にそって「共通の情報」を「取扱商品」とすると
  - 書籍 は 取扱商品(の一種)です
  - 家電 は 取扱商品(の一種)です

- このように「共通であるし、共通であるべき要素」を1箇所にまとめると保守がしやすい、というのを説明して「継承が必要になる事が理解できる」土壌を作っていきます
  - ここで本来的には「たまたま同じもの、は含まない」「〃共通であるべきもの〃だけを切り出す」必要があるのですが、その辺は難易度が高いのでいったん説明はオミットしてもよいかと思います
- この辺の「切り出しができる」は設計レベルのスキルなので「まだ先の話」ですが、こういった理由があるから「継承ってのを使う事があるから書式くらいは覚えておきましょう」という流れで継承の文法を教えるとよいと思います
- protected、オーバーライド(上書き)についての「書き方」と「基本的な挙動」を一緒に学習してもらいましょう

has-a関係とTrait

- is-a関係が出ているので、対としてhas-a関係についてこのタイミングで学んでもらうのはよいと思います
  - ただ、学習指針によっては「今はまだTraitを学ばせるつもりはない」といった範囲の問題もあるかと思うので、その辺りは柔軟に
- has-a関係のように「部品として何かを実装したい」時の方法の一つとしてTraitがあるので、Traitも「文法」くらいはおさえておくとよいでしょう
  - 特に「classの中でuseで使う」あたりは覚えておくとコードを読む時にも役に立ちます(というか知らないと読めないコードが出てきます)

インタフェース

- これも色々な側面があるのですが、実務的には「入れ替えがきく」という辺りがメリットに直結するのではないかと思うので、その辺りを基準に説明をしていきましょう
- まず「前提になる認識」から学習していきましょう

ユニットバスの規格など

- ユニットバスの縦横高さには、実は規格があります
  - 1216 (イチニイイチロク)
    - 内径の幅120cm × 内径の奥行き160cm
  - 1616 (イチロクイチロク)
  - 1618 (イチロクイチハチ)
  - 1620 (イチロクニイマル)
  - 等
- 「同じ規格」でも様々な機能やデザインのものがある
  - 割とゆっくり悩む事ができる
  - 後でリフォームとかで入れ替える事もできる
- 似たようなお話として「ネジ」「電池」などがあります

規格が決まっていると

- 色々なメーカや人が「色々なモノ」をその規格に沿って作る
- 買い手は「規格が同じ」であれば(基本)「色々なモノ」からチョイスが出来るし、気に入らなければ入れ替える事もできる

- これが「規格が決まっている」事の利点です
  - もちろんその分「自由が奪われている」と見なすことも出来ます
  - この辺は「既製服(レディメイド)」と「オートクチュール(お仕立て)」の差、とかとも似ていますね

で、インタフェース

- プログラム(を使う側)にとって重要なのは以下の2点
  - どんな入力を受け付けて
  - その結果、どんな出力を返してくるのか
- この2つが決まっていたら、内部の実装は(使う側としては)わりと「気にしなくても良い」事柄になります
- 例題として「奇数偶数判定」を取り上げてみましょう
- 引数は「整数1つ」、戻り値は「奇数ならtrue、偶数ならfalse」としておきます
  - なので、関数名は「isOdd()」としておきます

```
function isOdd(int $num): bool
{
    //「2で割ったあまり」が1なら奇数、0なら偶数
    return ($num % 2) === 1;
}

function isOdd(int $num): bool
{
    //「2で割って端数切り捨てして2倍する」値が元の値と異なったら奇数、一緒なら偶数
    return (((int)($num / 2)) * 2) !== $num;
}

function isOdd(int $num): bool
{
    // 1bit目が立っていたら奇数、寝ていたら偶数
    return ($num & 1) === 1;
}
```

- いずれも「引数として整数を受け取る」「処理として〝奇数か偶数か〟の判定をする」「結果として、奇数ならtrue、偶数ならfalseを返す」のは同じです
  - また、(意図的にですが)関数名も同じです
- このようにして「引数、関数(メソッド)名、戻り値、期待する動き」を「規格」にする事ができます
- この「規格」を定義するのがインタフェースになります……という流れから「インタフェースの定義」「継承して実装」を説明していくとよいでしょう

この後は……

- 静的メソッド(プロパティ)、クラスの抽象化(abstract)、マジックメソッド、無名クラス等といった「文法」
  - この辺から「PHP8で新しく追加された文法」の学習が本格化します
- クラスのオートローディング、オブジェクトのcloneやシリアライズ、参照といった「文法と知識にまたがるもの」
- DI(Dependency Injection)やSOLID、GoFのデザインパターンといった「知識」
- クラスの切り方など「設計」

- 等など、まだまだ道半ばですが、焦らずに教えていく事が大事かと思います
  - 優先順位は「自分(現場)がよく使う知識」から、でよいかと思います

- 技術は常にそうだけど「こんな困り事がある」からその技術が生まれる事がおおいので。そこを押さえると学習がはかどりやすいと思います

- 「いまある道具はなぜこの形をしているのか?」
  - UNIX MAGAZINE Classic

まとめ

- わからないうちは「どう学んだらいいかわからない」のですが、理解すると次は「どう教えたらいいかわからない」になりやすいと思います
- ただ、クラスはおそらく現在「知らないと現場で働けない」レベルで必須だと思います
- しかしまた一方で「学ぶべき量が多い」のも事実です
- その辺の交通整理の一助になれば幸いです

PHP技術者認定機構