PHP 練習問題．07 クラスとオブジェクトの解説

（参考）
※　この資料では、クラスと実行部分を一緒に記載したり、複数のクラスをひとつの PHP のファイルに記載したりしていますが、一般的には、1 クラスにつき 1 ファイルにします。
※　クラスのファイル名はクラスと同じ名前にします。
※　別のファイルにしたクラスのファイルは、require_once()で読み込みます。

(1)　オブジェクトって何？！

オブジェクトとは「もの」です。
プログラミングでいうと、

変数、定数、関数、クラス、インスタンス、データベース、テーブル、レコード、カラム

すべてのものが「オブジェクト」です。

それらの「オブジェクト」が、

- どのような関係なのか
- どのような関連性があるのか

を考えるのが「オブジェクト指向」と呼ばれるものです。

プログラム言語では、

関係性の高いものを集めてひとまとめにした「もの」の設計図

のことを「クラス」といいます。

クラスは次のように定義します。

```
<?php

/**
 * サンプルクラス 1
 */
class SampleClass1
{
    // クラス名は、パスカルケース（先頭大文字で単語の区切りを大文字）にするのが一般的です。
    // クラスとクラスのメンバ（プロパティ、定数、メソッドなど）は、「PHPDoc」を使って使い方の説明を書きましょう！

    // クラスの中には、メンバを作成することができます。
    // クラスのメンバは、定数、変数（プロパティ）、メソッド、です。

    // クラス定数は、const というキーワードを使って定義します。
    // 定数名は、大文字のスネークケース（単語の区切りに_を使う）にするのが一般的です。

    /** @var string サンプルの定数 */
    public const SAMPLE_CONSTANT = 'サンプルの定数';

    // クラス変数（プロパティといいます）
    // 変数名は、小文字のスネークケース（単語の区切りに_を使う）にするのが一般的です。
```

```
    /** @var string サンプルの変数 */
    public $sample_var = 'サンプルの変数';

    // クラスメソッド
    // メソッド名は、キャメルケース（先頭小文字で単語の区切りを大文字）にするのが一般的です。

    /**
     * サンプルのメソッド
     *
     * @return string サンプルの変数
     */
    public function sampleFunction()
    {
        // クラスの中でメンバにアクセスするときは、
        // $this という変数（疑似変数）とアロー演算子（->）を使います。
        return $this->sample_var;
    }
}

// クラスを使うときは、new というキーワードを使って、クラスのインスタンス（実体）を作ります。
$obj = new SampleClass1();

// インスタンス化されたクラスのメンバにアクセスするときは、「->」（アロー演算子）を使います。
// Java や JavaScript など、他の言語では「.」を使いますね。
echo $obj->sampleFunction();
```

```
echo '<br>';

// public なプロパティは値を書き換えることができます。
$obj->sample_var = '今日はラーメンが食べたい';
echo $obj->sampleFunction();

echo '<br>';

// 定数は、「クラス名::定数名」でアクセスできます。
echo SampleClass1::SAMPLE_CONSTANT;

// public であっても、定数は書き換えることができません。
// SampleClass1::SAMPLE_CONSTANT = '書き換えできません';

echo '<br>';

// 別のインスタンスのプロパティの値は、別の値になります。
$obj1 = new SampleClass1();
$obj2 = new SampleClass1();

$obj1->sample_var = 'あいうえお';
$obj2->sample_var = 'かきくけこ';

echo $obj1->sampleFunction();
echo '<br>';
echo $obj2->sampleFunction();
```

(2)　public と private とコンストラクタ

クラスのメンバには、「どこからアクセスできるか」という「アクセス修飾子」というものがあります。
「アクセス修飾子」には

public、private、protected

があります。それぞれどのようになっているかというと、

- public => クラスの中、外からでも、自由にアクセスできる。
- private => 同じクラスの中でしかアクセスできない。
- protected => 同じクラスと、クラスを継承したサブクラスでしかアクセスできない。

クラスのインスタンスを作ったときに自動的に動くメソッドのことを

コンストラクタ

といいます。
Java のコンストラクタはクラス名と同じ名前にしますが、PHP の場合は、「__consruct()」を使います。

```php
<?php

/**
 * サンプルクラス 2
 */
class SampleClass2
{
    // public キーワードを使うと、クラスの外からでもアクセスできます。

    /** @var string public なプロパティ */
    public $public_var;

    // private キーワードを使うと、クラスの外からはアクセスできません。

    /** @var string private なプロパティ */
    private $private_var;

    /**
     * コンストラクタ
     */
    public function __construct()
    {
        // コンストラクタでプロパティを初期化します。
        $this->public_var = 'こんにちは！';
        $this->private_var = '毎度お世話になります！';
    }
```

```
    // Java と違って、メソッドをオーバーロード（引数の数が違う同じ名前のメソッドを作成すること）はできません。

    /**
     * コンストラクタ
     */
    // public function __construct($str1, $str2)
    // {
    //     $this->public_var = $str1;
    //     $this->private_var = $str2;
    // }

    // ちなみに、「デストラクタ」といって、
    // インスタンスが破棄されるときに動くメソッドもあるのですが、
    // PHP の場合はリクエスト→レスポンスが終わると自動的にオブジェクトが破棄されるので、
    // あまり使い所はないかもしれません。

    /**
     * デストラクタ
     */
    public function __destruct()
    {
        // インスタンスが破棄されるときに行いたい処理を書く。
    }
}

$obj = new SampleClass2();
echo $obj->public_var;
```

```
// private なプロパティにクラスの外からアクセスすることはできません。
$obj->private_var = 'こんばんは！';    // エラーになります。
echo $obj->private_var;
```

(3)　カプセル化

クラスの外部からアクセスしてほしくないプロパティやメソッドを隠すことを

カプセル化

といいます。
カプセル化するには、プロパティやメソッドを private にします。

```
<?php

/**
 * サンプルクラス 3
 */
class SampleClass3
{
    // private なプロパティを作ります。

    /** @var string プライベートな変数 */
    private $private_var;

    // private なプロパティにクラスの外部から値を代入するには、
    // プロパティに値を代入するためのメソッドが必要です。

    // コンストラクタを利用する場合、コンストラクタに引数を与えます。
```

```
/**
 * コンストラクタ
 *
 * @param string $private_var
 */
public function __construct($private_var)
{
    // プロパティをコンストラクタの引数の値で初期化します。
    $this->private_var = $private_var;
}

// プロパティに値を設定するメソッドを作ります。
// プロパティに値をセットするので、「セッター」といいます。

/**
 * プライベートな変数に値を設定します。
 *
 * @param string $private_var
 * @return void
 */
public function setPrivateVar($private_var)
{
    $this->private_var = $private_var;
}

// private なプロパティの値をクラスの外部から読み取るには、
// プロパティの値を返却するメソッドが必要です。
```

```
    // プロパティから値をゲットしてくるので、「ゲッター」といいます。

    /**
     * プライベートなプロパティの値を取得します。
     *
     * @return string プライベートなプロパティの値
     */
    public function getPrivateVar()
    {
        return $this->private_var;
    }
}

// インスタンスを作るときに、private なプロパティに値を代入してみます。
$obj = new SampleClass3('こんにちは！');

// private なプロパティの値を直接取得すると怒られます。
echo $obj->private_var;    // エラーになります。

// private なプロパティの値を取得するメソッドを使ってみます。
echo $obj->getPrivateVar();

echo '<br>';

// プロパティに値を設定するメソッドを使ってみます。
$obj->setPrivateVar('まいどお世話になります！');
```

```
echo $obj->getPrivateVar();
```

(4)　static なメンバと static でないメンバ

static なプロパティは、クラス全体で (インスタンスが別であっても) 共用されます。

```
<?php

/**
 * サンプルクラス 4
 */
class SampleClass4
{
    // static なメンバには「static」というキーワードを使います。

    /** @var int static なプロパティ */
    public static $num1 = 0;

    // static でないプロパティは、インスタンスごとに別のものとして扱われます。

    /** @var int static でないプロパティ */
    public $num2 = 0;

    // static と static でないプロパティの考え方は、Java と同じです。

    /**
     * コンストラクタ
     */
```

```
    public function __construct()
    {
        // 同じクラス内の static なプロパティを使うときは、self::で呼び出します。
        self::$num1++;

        // 同じクラス内の static でないプロパティを使うときは、$this->で呼び出します。
        $this->num2++;
    }
}

// static なプロパティと static でないプロパティを比べてみます。
echo '1 回目<br>';
$obj = new SampleClass4();
echo SampleClass4::$num1 . '<br>';
echo $obj->num2 . '<br>';

echo '<br>==========<br>';

echo '2 回目<br>';
$obj = new SampleClass4();
echo SampleClass4::$num1 . '<br>';
echo $obj->num2 . '<br>';

echo '<br>==========<br>';

echo '3 回目<br>';
$obj = new SampleClass4();
```

```
echo SampleClass4::$num1 . '<br>';
echo $obj->num2 . '<br>';

echo '<br>==========<br>';

echo '4 回目<br>';
$obj = new SampleClass4();
echo SampleClass4::$num1 . '<br>';
echo $obj->num2 . '<br>';

echo '<br>==========<br>';

// static でないなプロパティは、インスタンスごとに別のものなので、
// 値はコンストラクタで毎回初期化されます。
// static なプロパティは、クラスで共用されるので、
// インスタンスが作られるたびにコンストラクタで 1 ずつ足されます。

// ちなみに、クラスのインスタンスを配列の要素に代入することも可能です。
$obj = null;    // 作成済みのインスタンスを破棄します。null を代入することで破棄できます。
for ($i = 0; $i < 4; $i++) {
    $obj[$i] = new SampleClass4();
}

foreach ($obj as $k => $v) {
    echo $k.'回目<br>';
    echo $v->num2 . '<br>';
    echo '<br>==========<br>';
```

```
}
```

(5)　クラスの継承・その１

クラスは、おおもとのクラスを継承したクラスを作成することができます。
おおもとのクラスのことを、

スーパークラス、ベースクラス、親クラス、基底クラス、

と言ったりします。
おおもとのクラスを継承したクラスは、

サブクラス、子クラス、派生クラス、

と言ったります。
クラスを継承するには「extends」というキーワードを使います。

```
<?php

/**
 * サンプルクラス 5
 */
class SampleClass5
{
    /** @var string こんにちは！ */
    public $var = 'こんにちは！';

    /**
     * プロパティの値を返却します。
     *
```

```
     * @return void
     */
    public function sample5()
    {
        return $this->var;
    }
}

/**
 * サンプルクラス 5 を継承したサンプルクラス 5 サブクラス
 */
class SampleClass5Sub extends SampleClass5
{

}

// サブクラスからスーパークラスのプロパティにアクセスすることができます。
// サブクラスのインスタンスを作ります。
$obj = new SampleClass5Sub();
echo $obj->var;

echo '<br>==========</br>';

// サブクラスからスーパークラスのメソッドも使えます。
echo $obj->sample5();
```

(6)　クラスの継承・その２

継承とプロパティとメソッドについて、更に詳しく見ていきます。

```
<?php

/**
 * サンプルクラス 6
 */
class SampleClass6
{
    /** @var string スーパークラスのプロパティ */
    protected $parent_var;

    /**
     * プロパティに値を設定します。
     *
     * @param string $parent_var
     * @return void
     */
    public function setVar(string $parent_var)
    {
        $this->parent_var = $parent_var;
    }

    /**
```

```
     * プロパティの値を返却します。
     *
     * @return void
     */
    public function getVar()
    {
        return $this->parent_var;
    }
}

/**
 * サンプルクラス 6 を継承したサンプルクラス 6 サブクラス
 */
class SampleClass6Sub extends SampleClass6
{
    // サブクラス独自のプロパティを作ります。

    /** @var string こんにちは！ */
    private $child_var = 'こんにちは！';

    /**
     * スーパークラスのプロパティの値を利用したサブクラスのメソッド
     *
     * @return string
     */
    public function getVar()
    {
```

```
        // スーパークラスのメソッドと同じ名前で再定義するときに親のメソッドを呼び出すには、
        // parent::を使います。
        // これを「オーバーライドする」と言います。
        return parent::getVar() . $this->child_var;
    }
}

$obj = new SampleClass6Sub();
$obj->setVar('みらいさん、');

// サブクラスで同じ名前でオーバーライドされているので、
// スーパークラスのメソッドが呼ばれることはありません。
echo $obj->getVar();
```

(7)　クラスの継承・その３

スーパークラスを継承したサブクラスに、コンストラクタを定義しないことがあります。
その場合でも、サブクラスのインスタンスを生成したときに、親クラスのコンストラクタが暗黙的に動きます。

```
<?php

/**
 * サンプルクラス 7
 */
class SampleClass7
{
    /** @var string サンプルのプロパティ */
    public $sample;

    /**
     * コンストラクタ
     */
    public function __construct()
    {
        // サンプルのプロパティを初期化します。
        $this->sample = 'こんにちは！';
    }
}

/**
```

```
 * サンプルクラス 7 を継承したサンプルクラス 7 サブクラス
 */
class SampleClass7Sub extends SampleClass7
{
    // サブクラスにはコンストラクタを定義しません。
}

$obj = new SampleClass7Sub();
// サブクラスにコンストラクタを定義していなくても、
// 暗黙的にスーパークラスのコンストラクタが動きます。
echo $obj->sample;
```

(8)　クラスの継承・その４

スーパークラスを継承したサブクラスにコンストラクタを定義しなかったとき、サブクラスのインスタンスを生成したら暗黙的にスーパークラスのコンストラクタが動きました。
ところが、サブクラスにコンストラクタを定義したときは、暗黙的にスーパークラスのコンストラクタが動くことはありません。

```
<?php

/**
 * サンプルクラス 8
 */
class SampleClass8
{
    /** @var string サンプルのプロパティ */
    public $sample;

    /**
     * コンストラクタ
     */
    public function __construct()
    {
        // サンプルのプロパティを初期化します。
        $this->sample = 'こんにちは！';
    }
}
```

```php
/**
 * サンプルクラス 8 を継承したサンプルクラス 8 サブクラス
 */
class SampleClass8Sub extends SampleClass8
{
    // サブクラスにもコンストラクタを定義します。

    public function __construct()
    {
        // 処理は何も書かないことにします。
    }
}

$obj = new SampleClass8Sub();
// サブクラスにコンストラクタを定義した場合は、
// 暗黙的にスーパークラスのコンストラクタが呼ばれることはありません。
// サブクラスのコンストラクタで何も処理をしていないので、$sample には何も代入されておらず、何も表示されません。
echo $obj->sample;
```

(9)　クラスの継承・その５

サブクラスのコンストラクタでスーパークラスのコンストラクタを明示的に呼び出してみます。

```
<?php

/**
 * サンプルクラス 9
 */
class SampleClass9
{
    /** @var string サンプルのプロパティ */
    public $sample;

    /**
     * コンストラクタ
     */
    public function __construct()
    {
        // サンプルのプロパティを初期化します。
        $this->sample = 'こんにちは！';
    }
}

/**
 * サンプルクラス 9 を継承したサンプルクラス 9 サブクラス
```

```
 */
class SampleClass9Sub extends SampleClass9
{
    // サブクラスにもコンストラクタを定義します。

    /**
     * コンストラクタ
     */
    public function __construct()
    {
        // スーパークラスのコンストラクタを呼び出したいときは、
        // 明示的にスーパークラスのコンストラクタを呼び出します。
        parent::__construct();
    }
}

$obj = new SampleClass9Sub();
// サブクラスのコンストラクタでスーパークラスのコンストラクタを明示的に呼び出したことで、
// サブクラスのコンストラクタでスーパークラスのコンストラクタも動きます。
echo $obj->sample;
```

(10) 抽象クラス

抽象クラスは、それ自体は中身は空っぽで、継承先のクラスで中身を実装するためのものです。
「こういうメソッドを作って実装してくださいね」という、クラスの「設計図」みたいなものです。
抽象クラスを作るには、「abstract」というキーワードを使います。

```
<?php

/**
 * サンプルの抽象クラス
 */
abstract class AudioClass
{
    // 抽象クラスのメソッドは、中身を実装しません。

    /**
     * 音楽を再生します。
     *
     * @return string
     */
    abstract protected function play();

    /**
     * 音楽を止めます。
     *
     * @return void
```

```
     */
    abstract protected function stop();
}

/**
 * サンプルの抽象クラスを継承したサンプルクラス
 */
// class MusicPlayer extends AudioClass
// {
//     // スーパークラスで定義したメソッドを実装しないとエラーになります。
// }

/**
 * サンプルの抽象クラスを継承したサンプルクラス
 */
class MusicPlayer extends AudioClass
{
    // 抽象クラスであるスーパークラスのメソッドを実装します。

    /**
     * 音楽を再生します。
     *
     * @return string
     */
    public function play()
    {
        return '音楽を再生します。';
```

```
    }

    /**
     * 音楽を止めます。
     *
     * @return void
     */
    public function stop()
    {
        return '音楽を止めます。';
    }
}

// 抽象クラスのインスタンスを生成しようとするとえらーになります。
// $obj = new AudioClass();

// 抽象クラスを継承したクラスは、通常通り、インスタンスを生成できます。
$obj = new MusicPlayer();
echo $obj->play();

echo '<br>==========</br>';

echo $obj->stop();
```

(11) インターフェイス

インターフェイスは、それ自体は中身は空っぽで、継承先のクラスで中身を実装します。
「こういうメソッドを作って実装してくださいね」という、クラスの「設計図」みたいなものです。
抽象クラスと違うところは、

- クラスは 1 つのクラスからしか継承できない
- インターフェイスは、複数のインターフェイスを継承してクラスを実装することができる

という点です。

インターフェイスは「interface」というキーワードを使って定義します。

```
<?php

/**
 * 音楽プレーヤーのインターフェイス
 */
interface AudioInterface
{
    // インターフェイスのメソッドは、中身を実装しません。
    // アクセス修飾子は public にしなければなりません。

    /**
     * 音楽を再生します。
     *
     * @return string
```

```
     */
    public function play();

    /**
     * 音楽を止めます。
     *
     * @return void
     */
    public function stop();
}

// もう一つインターフェイスを作ります。

/**
 * 電話機のインターフェイス
 */
interface PhoneInterface
{
    /**
     * 電話をかけます。
     *
     * @return string
     */
    public function call();

    /**
     * 電話を切ります。
```

```
     *
     * @return string
     */
    public function hungUp();
}

// 2つのインターフェイスを実装するクラスを作ります。
// インターフェイスを継承する場合は、「implements」というキーワードを使います。

/**
 * スマートフォンクラス
 */
class SmartPhone implements AudioInterface, PhoneInterface
{
    /**
     * 音楽を再生します。
     *
     * @return string
     */
    public function play()
    {
        return '音楽を再生します。';
    }

    /**
     * 音楽を止めます。
     *
```

```
     * @return void
     */
    public function stop()
    {
        return '音楽を止めました。';
    }

    /**
     * 電話をかけます。
     *
     * @return string
     */
    public function call()
    {
        return '電話をかけます。';
    }

    /**
     * 電話を切ります。
     *
     * @return string
     */
    public function hungUp()
    {
        return '電話を切りました。';
    }
}
```

```php
// 音楽プレーヤーのインターフェイスと電話機のインターフェイスを実装したスマートフォンクラスを使います。
$smart_phone = new SmartPhone();

echo $smart_phone->play();

echo '<br>==========</br>';

echo $smart_phone->stop();

echo '<br>==========</br>';

echo $smart_phone->call();

echo '<br>==========</br>';

echo $smart_phone->hungUp();
```