■PHP クラスとオブジェクト （オブジェクト指向プログラミング：OOPの補足資料）

作成：2010.10.12/2015.10.15修正　yoshi

【1】クラスとは　　**→実体を作るための設計図（処理とプロパティ）のようなもの**

●クラスの構成要素

以下の要素から構成されている。クラスのメンバーと呼ばれる

◆プロパティ　　クラスに属する変数で、メソッド外で定義する変数

◆メソッド　　クラスに属する処理（処理手順、関数）

◆クラスの構成順　　※クラス定義は、「**class**」キーワードを先頭に付けて、クラス名が続く

class <クラス名> {
　プロパティ（フィールド）
　コンストラクタ ⟶
　（デストラクタ）
　メソッド
}

```
function __construct (<引数>) {
    コンストラクタで行う処理
}
```

注）：**__construct** を使わず、クラス名と同じの場合もコンストラクタ処理

注）：デストラクタは、function **__destruct** (){ … } の関数記述で、クラスの終了時に実行されるメソッド

●インスタンスは、　**→クラスから作成される実体そのもの**

・クラスから「**new**」キーワードを利用して、作成する

・クラスのプロパティへのアクセス… **<インスタンス変数名>->プロパティ名>**

・メソッドの呼び出し（アクセス）… **<インスタンス変数名>->メソッド名>()**　（メソッドには、引数と戻り値がある）

・オブジェクトは、広義の意味で利用されるが、インスタンスと同じ意味。

・インスタンスの変数は、参照（リファレンス）型である

●コンストラクタとは、

インスタンスの「初期化処理」を記述したもので、「**new**」キーワードでインスタンスを作成するタイミングで自動的に呼び出される（初期化用の特殊なメソッド）

●メソッドのオーバーロード

同じクラスに、同じ名前で引数の型が異なるメソッドを定義すること（シグネチャ）

注）：メソッドのシグネチャとは、メソッドの名前と引数の並びのこと。メソッドが同じで、引数が違うものは定義できる。

●「->」オブジェクト演算子 … クラス内の下位階層のメソッド又はプロパティの指定

「インスタンス名->関数名();」　　… **メソッド**：　クラス内の関数呼び出し（指定）

「インスタンス名->変数;」　　… **プロパティ**：クラス内の変数呼び出し（指定）

●コンストラクタのthis

コンストラクタから別のコンストラクタの呼び出し

●this演算子（擬似変数） … クラス内の関数でクラス内変数へのアクセス

「**$this->クラス内変数**」のようにオブジェクト演算子「->」を使って、下位の階層を指示する記述にする

●「::」スコープ定義演算子

オブジェクトを作成して変数名（プロパティ）を指定しなくても、「**クラス名::変数名**」のように直接呼び出すことができる

キーワード	内容
self	自分自身を意味する
parent	親クラスを意味する
static	静的クラスを意味する

→「**self::<メソッド>**」のように記述

【2】継承（インヘリタンス）　　inheritance

あるクラスの機能を他のクラスが引き継ぐ仕組み

似ている少し違うクラスを作成する場合、同じ記述のところを省略して、違う部分だけを記述すること

class <クラス名> extends <スーパークラス名> { … }
継承するクラス↑　　継承元↑

●メソッドのオーバーライド　　override

スーパークラスと同じメソッドを再定義すること

superのキーワードで親クラスの参照を表す暗黙的な変数

final修飾子は、オーバーライド禁止

private修飾子は、オーバーライドできない

コンストラクタは、privateなメンバーで継承されない

オーバーライドされた元のメソッドや静的プロパティにアクセスするには、「**parent::**」で参照します。

注）：派生クラスとは、子クラスや孫クラスをまとめて元のクラスから派生したクラスという意味

●ラッパークラス　　wrapper class

基本データ型を包んでオブジェクトとして扱う。

●セッタ、ゲッタ（アクセサとも呼ぶ）

クラス内のprivate変数にアクセスするためのメソッドのことを「アクセサメソッド」と呼ぶ

フィールド名とセッタ、ゲッタ→フィールドに代入するメソッド

セッタ、ゲッタで取得、設定する値は、プロパティと呼ぶ

●カプセル化
・修飾子としての役割は、クラス内部の機能をクラス外部から隠蔽することが目的で使用される
・フィールドやメソッドの繋ぎの実装を1箇所にまとめることをカプセル化と呼び、クラスの外部からは
フィールドを直接操作できないようにして、必ずメソッドを経由してフィールドを間接的に操作すること
・カプセル化はクラス、フィールド、メソッドに適切な「**アクセス修飾子**」を付けることで実現。
◆アクセス修飾子は、クラス内のメンバー変数(プロパティ)やメンバー関数(メソッド)に対するアクセスの可否をきめる
・アクセス修飾子には、public、protected、指定なし、private などある

アクセス修飾子	概要
public	どこからでもアクセス可能(デフォルト)
protected	現在のクラスとサブクラスのみでアクセス可能
private	現在のクラス内部でのみアクセス可能

←省略した場合は、
メソッド、プロパティは**public** 扱い

※PHPは、クラスの多重継承ができないので「インターフェース」を活用して、多重継承と似たような効果を実現できる
(多重継承不可:サブクラスが同時に複数のスーパークラスを継承できないことを言う)

【3】ポリモーフィズム　　Polymorphism
プログラミング言語の各要素(定数、変数、式、オブジェクト、関数、メソッドなど)について、それらが複数の型に属することを許すという性質を指す。多態性、多相性、多様性とも呼ばれる。

【3-1】インターフェイス
インターフェイスで定義するメソッドは、抽象メソッド(抽象データ型)といわれ、シグネチャだけで中身がないメソッドである
(注:クラスを具象データ型と呼ぶ)
・クラスの使い方を定義したもの

```
interface 〈インターフェイス名〉{
        〈メソッド1のシグネチャ〉
        〈メソッド2のシグネチャ〉
}
```

・インターフェイスの実装宣言

```
class 〈クラス名〉 implements 〈インターフェイス名〉{
        〈メソッド1の実装〉
        〈メソッド2の実装〉
}
```

←〈インターフェース名〉内をコンマ区切りで、複数定義が可能。
インターフェースを実装したクラスは、インターフェースで宣言されているメソッドを全て定義する必要がある

変数に代入されている「インスタンスの型」に応じて、実際に呼び出せる
メソッドが変わる(動的デスパッチ)→「同じ操作で異なる動作をさせること」をポリモーフィズム(多相性)→
変数の型でなく、代入しているインスタンス型(**instanceof** 演算子)に応じて呼ばれるメソッドが切り替わる。

※インターフェイスもクラスのように「**extends** 」で継承でき、しかも多重継承ができる

【4】例外(Exception)　　プログラムを実行する最中に発生するエラー
・例外が発生するとメソッド内のそれ以降の処理を中断して、呼び出し元のメソッドに処理を戻す→「例外がスローされる」という。
・例外をキャッチするステートメント:**try～catch**
・try は、例外を発生させる可能性のある処理を呼び出す場合に用います。
・try { … } の間で例外が発生した場合、catch を用いてこの例外を捕捉します。

●NullPointerException
参照型変数には、nullリテラルを代入できる(nullは、インスタンスを参照していないことを示すリテラル)
この状態の変数に対してアクセスすると、NullPointerExceptionが発生する

参照型変数:インスタンスに代入した場合は、インスタンスの値がコピーされるのではなく参照がコピーされる。

【5】_autoload関数
・未定義のクラス/インターフェイスを使用しようとした時に 自動的に呼び出される特別な関数であり、
呼び出した未定義のクラス名が引数として渡されるもの。それ自体なんら実装を持たない名前だけ予約された関数
・_autoload関数を利用すると、クラスごとにrequire_once関数を呼び出す必要がなくなるので、コーディングがシンプルになり、
クラスが呼び出されたタイミングで自動的に同名のクラスファイルをincludeするような用途で使用する
・「クラス名.class.php」のような形式であらかじめファイル名に一定の規則性を設けた上で、_autoload関数内で
require_once関数を呼び出す。
・**spl_autoload_register** を推奨
→**spl_autoload_registe** r を使えば、より柔軟にクラスのオートロードができます。
… 指定した関数を _autoload() の実装として登録する。autoload 関数のキューを作成し、定義された順にそれを実行していきます。一方 _autoload() は、一度しか定義できません。

```
function my_autoloader($class) {
    include 'classes/' . $class . '.class.php';
}
```

注):SPLは、標準的な問題を解決するためのインターフェイスやクラスを集めたものです。

(注):ネーミングルール例→　**クラス名**:大文字で始まる単語
変数名:小文字で始まる単語
単語は、**キャメル形式**