01. アクセス修飾子

◇ static

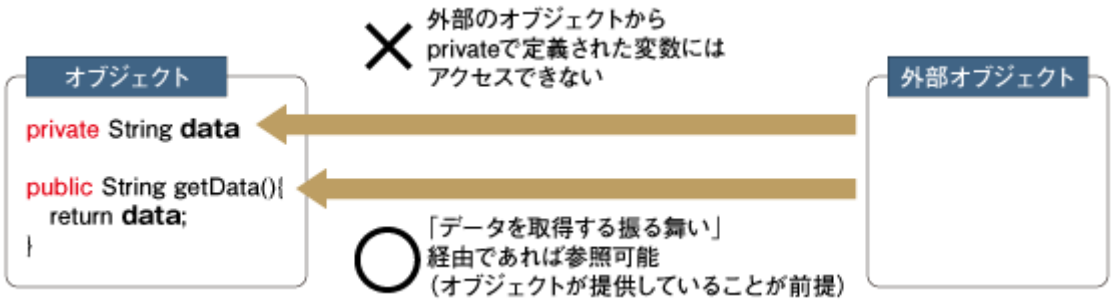
別ファイルでのメソッドの呼び出しにはインスタンス化が必要である。しかし、static修飾子をつけることで、インスタンス化しなくとも呼び出せる。生成されたオブジェクト自身から取り出す必要がなく、静的（オブジェクトの状態とは無関係）な、プロパティやメソッドに用いる。

◇ private

同じオブジェクト内でのみ呼び出せる。

- **Encapsulation（カプセル化）**

カプセル化とは、システムの実装方法を外部から隠すこと。オブジェクト内のプロパティにアクセスするには、直接データを扱う事はできず、オブジェクト内のメソッドを呼び出して、アクセスしなければならない。



◇ protected

同じクラス内と、その親クラスまたは子クラスでのみ呼び出せる。

◇ public

どのオブジェクトでも呼び出せる。

02. メソッド

◇ メソッドの実装手順

1. その会社のシステムで使われているライブラリ
2. phpのデフォルト関数（引用：php関数リファレンス，https://www.php.net/manual/ja/funcref.php）
3. 新しいライブラリ

◇ 値を取得するアクセサメソッドの実装

Getterでは、プロパティを取得するだけではなく、何かしらの処理を加えたうえで取得すること。

【実装例】

- **Getter**

```
class ABC {

    private $property;

    public function getEditProperty()
    {
        // 単なるGetterではなく、例外処理も加える。
        if(!isset($this->property){
            throw new ErrorException('プロパティに値がセットされていません。')
        }
        return $this->property;
    }

}
```

◇ 値を設定するアクセサメソッドの実装

- **Setter**

『Mutable』なオブジェクトを実現できる。

【実装例】

```
class Test01 {

    private $property01;

    // Setterで$property01に値を設定
    public function setProperty($property01)
    {
        $this->property01 = $property01;
    }

}
```

- **マジックメソッドの `__construct()`**

Setterを持たせずに、`__construct()`だけを持たせれば、ValueObjectのような、『Immutable』なオブジェクトを実現できる。

【実装例】

```
class Test02 {

    private $property02;

    // コンストラクタで$property02に値を設定
    public function __construct($property02)
    {
        $this->property02 = $property02;
    }

}
```

- **『Mutable』と『Immutable』を実現できる理由**

Test01クラスインスタンスの`$property01`に値を設定するためには、インスタンスからSetterを呼び出す。Setterは何度でも呼び出せ、その度にプロパティの値を上書きできる。

```
$test01 = new Test01

$test01->setProperty01("プロパティ01の値")

$test01->setProperty01("新しいプロパティ01の値")
```

一方で、Test02クラスインスタンスの`$property02`に値を設定するためには、インスタンスを作り直さなければならない。つまり、以前に作ったインスタンスの`$property02`の値は上書きできない。Setterを持たせずに、`__construct()`だけを持たせれば、『Immutable』なオブジェクトとなる。

```
$test02 = new Test02("プロパティ02の値")

$test02 = new Test02("新しいプロパティ02の値")
```

◇ メソッドチェーン

以下のような、オブジェクトAを最外層とした関係が存在しているとする。

【オブジェクトA（オブジェクトBをプロパティに持つ）】

```
class Obj_A{
    private $ObjB;

    public function getObjB()
    {
        return $this->ObjB;
    }
}
```

【オブジェクトB（オブジェクトCをプロパティに持つ）】

```
class Obj_B{
    private $ObjC;

    public function getObjC()
    {
        return $this->ObjC;
    }
}
```

【オブジェクトC（オブジェクトDをプロパティに持つ）】

```
class Obj_C{
    private $ObjD;

    public function getObjD()
    {
        return $this->ObjD;
    }
}
```

以下のように、返り値のオブジェクトを用いて、より深い層に連続してアクセスしていく場合...

```
$ObjA = new Obj_A;

$ObjB = $ObjA->getObjB();

$ObjC = $B->getObjB();

$ObjD = $C->getObjD();
```

以下のように、メソッドチェーンという書き方が可能。

```
$D = getObjB()->getObjC()->getObjC();

// $D には ObjD が格納されている。
```

◇ マジックメソッド

オブジェクトに対して特定の操作が行われた時に自動的に呼ばれる特殊なメソッドのこと。処理内容は自身で実装する必要がある。

- `__construct()`

クラスがインスタンス化される時に呼び出される。

- `__get()`

定義されていないプロパティや、アクセス権のないプロパティを取得しようとした時に、自動的に呼び出される。

```
class Example
{

    private $example = [];

    // 定義されていないプロパティが設定された場合、$exampleに連想配列として設定。
    public function __get($name)
    {
        return $this->example[$name];
    }

}
```

```
// 存在しないプロパティを取得。
$example = new Example();
$example->hoge;
```

- `__set()`

定義されていないプロパティや、アクセス権のないプロパティに値を設定しようとした時に、自動的に呼び出される。

```
class Example
{

    private $example = [];

    //
    public function __set($name, $value)
    {
        $this->example[$name] = $value;
    }

}
```

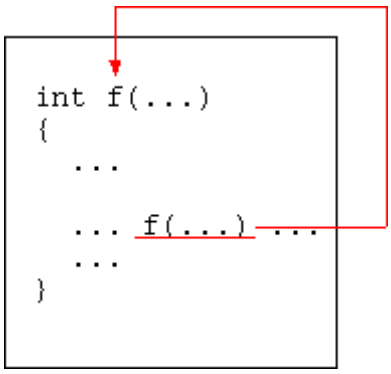
```
// 存在しないプロパティに値をセット。
$example = new Example();
$example->huga = 'aaa';
```

◇ Recursive call：再帰的プログラム

自プログラムから、自身自身を呼び出して実行できるプログラムのこと。

【具体例】

ある関数 `f` の定義の中に `f` 自身を呼び出している箇所がある。



◇ 高階関数とClosure（無名関数）

関数を引数として受け取ったり、関数自体を返したりする関数のこと。

- **無名関数を用いない場合**

【実装例】

```
## 第一引数のみの場合

// 高階関数を定義
function test($callback)
{
    echo $callback();
}

// コールバックを定義
// 関数の中で呼び出されるため、「後で呼び出される」という意味合いから、コールバック関数といえる。
function callbackMethod()
{
    return "出力成功";
}

// 高階関数の引数として、コールバック関数を渡す
test("callbackMethod");

// 出力結果
出力成功
```

```
## 第一引数と第二引数の場合

// 高階関数を定義
public function higher-order($param, $callback)
{
    return $callback($param);
}

// コールバック関数を定義
public function callbackMethod($param)
{
    return $param."の出力成功";
}
```

```
// 高階関数の第一引数にコールバック関数の引数、第二引数にコールバック関数を渡す
higher-order("第一引数", "callbackMethod");

// 出力結果
第一引数の出力成功
```

- **無名関数を用いる場合**

【実装例】

```
// 高階関数のように、関数を引数として渡す。
public function higher-order($param, $callback)
{
    $parentVar = "&親メソッドのスコープの変数"
    return $callback($param)
}

// 第二引数の無名関数。関数の中で呼び出されるため、「後で呼び出される」という意味合いから、コールバック関数といえる。
// コールバック関数は再利用されないため、名前をつけずに無名関数とすることが多い。
// 親メソッドのスコープの変数を引数として用いることができる。
high-order(第一引数,
        function($param) use($parentVar)
        {
            return $param.$parentVar."の出力成功";
        }
    )

// 出力結果
第一引数&親メソッドのスコープの変数の出力成功
```

03. データ型

プログラムを書く際にはどのような処理を行うのかを事前に考え、その処理にとって最適なデータ構造で記述する必要がある。そのためにも、それぞれのデータ構造の特徴（長所、短所）を知っておくことが重要である。

◇ Array型

- **多次元配列**

中に配列をもつ配列のこと。配列の入れ子構造が２段の場合、『二次元配列』と呼ぶ。

```
Array
(
    [0] => Array
        (
            [0] => リンゴ
            [1] => イチゴ
            [2] => トマト
        )

    [1] => Array
        (
```

```
            [0] => メロン
            [1] => キュウリ
            [2] => ピーマン
        )
)
```

- **連想配列**

中に配列をもち、キーに名前がついている（赤、緑、黄、果物、野菜）ような配列のこと。下の例は、二次元配列かつ連想配列である。

```
Array
(
    [赤] => Array
        (
            [果物] => リンゴ
            [果物] => イチゴ
            [野菜] => トマト
        )

    [緑] => Array
        (
            [果物] => メロン
            [野菜] => キュウリ
            [野菜] => ピーマン
        )
)
```

◇ Object型

プロパティ名とその値は、連想配列になっている。

```
Fruit Object
(
    [id:private] => 1
    [name:private] => リンゴ
    [price:private] => 100
)
```

04. TRUE vs. FALSE

◇ FALSE の定義

- **表示なし**
- **キーワード FALSE false**
- **整数 0**
- **浮動小数点 0.0**
- **空の文字列 " "**

- **空の文字列 ' '**
- **文字列 "0"（文字列としての0）**
- **要素数が 0 の配列$ary = array();**
- **プロパティーやメソッドを含まない空のオブジェクト**
- **NULL値**

◇ TRUE の定義

上記の値以外は、全て TRUE

◇ 変数に値が入っているのかを確かめるシリーズ

値	if($var)	isset	empty	is_null
$var=1	**true**	**true**	*false*	*false*
$var="";	*false*	**true**	**true**	*false*
$var="0";	*false*	**true**	**true**	*false*
$var=0;	*false*	**true**	**true**	*false*
$var=NULL;	*false*	*false*	**true**	**true**
$var	*false*	*false*	**true**	**true**
$var=array()	*false*	**true**	**true**	*false*
$var=array(1)	**true**	**true**	*false*	*false*

```
# 右辺には、上記に当てはまらない状態『TRUE』が置かれている。
if($this->$var == TRUE){
    処理A;
}

# ただし、基本的に右辺は省略すべき。

if($this->$var){
    処理A;
}
```

05. 条件式の実装方法

◇ 『else』はできるだけ用いない

- **『else』を用いる場合**

冗長になってしまう。

```
// マジックナンバーを使わずに、定数として定義
const noOptionItem = 0;

// RouteEntityからoptionsオブジェクトに格納されるoptionオブジェクト配列を取り出す。
if(!empty($routeEntity->options) {
    foreach ($routeEntity->options as $option) {

        // if文を通過した場合、メソッドの返り値が格納される。通過しない場合、定数が格納される。
        if ($option->isOptionItemA()) {
            $result['optionItemA'] = $option->optionItemA();
        } else {
            $result['optionItemA'] = noOptionItem;
            }

        if ($option->isOptionItemB()) {
            $result['optionItemB'] = $option->optionItemB();
        } else {
            $result['optionItemB'] = noOptionItem;
            }

        if ($option->isOptionItemC()) {
            $result['optionItemC'] = $option->optionItemC();
        } else {
            $result['optionItemC'] = noOptionItem;
            }
    };
}

return $result;
```

- **初期値と上書きのロジックを用いる場合**

よりすっきりした書き方になる。

```
// マジックナンバーを使わずに、定数として定義
const noOptionItem = 0;

// 初期値0を設定
$result['optionItemA'] = noOptionItem;
$result['optionItemB'] = noOptionItem;
$result['optionItemC'] = noOptionItem;

// RouteEntityからoptionsオブジェクトに格納されるoptionオブジェクト配列を取り出す。
if(!empty($routeEntity->options) {
    foreach ($routeEntity->options as $option) {
```

```
        // if文を通過した場合、メソッドの返り値によって初期値0が上書きされる。通過しない場合、初期値0が用いられる。
        if ($option->isOptionItemA()) {
            $result['optionItemA'] = $option->optionItemA();
        }

        if ($option->isOptionItemB()) {
            $result['optionItemB'] = $option->optionItemB();
        }

        if ($option->isOptionItemC()) {
            $result['optionItemC'] = $option->optionItemC();
        }
    };
}

return $result;
```

◇ エラー文

エラー文は、『ログファイル』に出力される。if文を通過してしまった理由は、empty()でTRUEが返ったためである。empty()がFALSEになるように、デバッグする。

```
if (empty($value)) {
    throw new Exception('Variable is empty');
}
return $value
```

06. 変数

◇ スーパーグローバル変数

スコープに関係なく、どのプログラムからでもアクセスできる連想配列変数

$GLOBALS (グローバル変数)	・グローバルスコープで使用可能なすべての変数への参照 ・連想配列として使用
$_SERVER (サーバー変数)	・サーバ情報および実行時の環境情報が格納される ・連想配列として使用
$_GET (ゲット変数)	・HTTP GET で送信された変数が格納される ・連想配列として使用
$_POST (ポスト変数)	・HTTP POST で送信された変数が格納される ・連想配列として使用
$_FILES (ファイル変数)	・HTTP POSTでアップロードされた変数が格納される ・連想配列として使用
$_REQUEST (リクエスト変数)	・HTTP リクエスト変数が格納される ・連想配列として使用
$_SESSION (セッション変数)	・セッション変数が格納される ・連想配列として使用
$_ENV (環境変数)	・環境変数が格納される ・連想配列として使用
$_COOKIE (クッキー変数)	・HTTP クッキー変数が格納される ・連想配列として使用

- **`$_SERVER` に格納されている値**

```
$_SERVER['SERVER_ADDR']            サーバのIPアドレス(例:192.168.0.1)
$_SERVER['SERVER_NAME']            サーバの名前(例:www.example.com)
$_SERVER['SERVER_PORT']            サーバのポート番号(例:80)
$_SERVER['SERVER_PROTOCOL']        サーバプロトコル(例:HTTP/1.1)
$_SERVER['SERVER_ADMIN']           サーバの管理者(例:root@localhost)
$_SERVER['SERVER_SIGNATURE']       サーバのシグニチャ(例:Apache/2.2.15...)
$_SERVER['SERVER_SOFTWARE']        サーバソフトウェア(例:Apache/2.2.15...)
$_SERVER['GATEWAY_INTERFACE']      CGIバージョン(例:CGI/1.1)
$_SERVER['DOCUMENT_ROOT']          ドキュメントルート(例:/var/www/html)
$_SERVER['PATH']                   環境変数PATHの値
(例:/sbin:/usr/sbin:/bin:/usr/bin)
$_SERVER['PATH_TRANSLATED']        スクリプトファイル名(例:/var/www/html/test.php)
$_SERVER['SCRIPT_FILENAME']        スクリプトファイル名(例:/var/www/html/test.php)
$_SERVER['REQUEST_URI']            リクエストのURI(例:/test.php)
$_SERVER['PHP_SELF']               PHPスクリプト名(例:/test.php)
$_SERVER['SCRIPT_NAME']            スクリプト名(例:/test.php)
$_SERVER['PATH_INFO']              URLの引数に指定されたパス名(例:/test.php/aaa)
$_SERVER['ORIG_PATH_INFO']         PHPで処理される前のPATH_INFO情報
$_SERVER['QUERY_STRING']           URLの?以降に記述された引数(例:q=123)
$_SERVER['REMOTE_ADDR']            クライアントのIPアドレス(例:192.168.0.123)
$_SERVER['REMOTE_HOST']            クライアント名(例:client32.example.com)
$_SERVER['REMOTE_PORT']            クライアントのポート番号(例:64799)
$_SERVER['REMOTE_USER']            クライアントのユーザ名(例:tanaka)
$_SERVER['REQUEST_METHOD']         リクエストメソッド(例:GET)
$_SERVER['REQUEST_TIME']           リクエストのタイムスタンプ(例:1351987425)
$_SERVER['REQUEST_TIME_FLOAT']     リクエストのタイムスタンプ(マイクロ秒)(PHP 5.1.0以降)
$_SERVER['REDIRECT_REMOTE_USER']   リダイレクトされた場合の認証ユーザ(例:tanaka)
$_SERVER['HTTP_ACCEPT']            リクエストのAccept:ヘッダの値(例:text/html)
$_SERVER['HTTP_ACCEPT_CHARSET']    リクエストのAccept-Charset:ヘッダの値(例:utf-8)
```

```
$_SERVER['HTTP_ACCEPT_ENCODING']  リクエストのAccept-Encoding:ヘッダの値(例:gzip)
$_SERVER['HTTP_ACCEPT_LANGUAGE']  リクエストのAccept-Language:ヘッダの値(ja,en-US)
$_SERVER['HTTP_CACHE_CONTROL']    リクエストのCache-Control:ヘッダの値(例:max-age=0)
$_SERVER['HTTP_CONNECTION']       リクエストのConnection:ヘッダの値(例:keep-alive)
$_SERVER['HTTP_HOST']             リクエストのHost:ヘッダの値(例:www.example.com)
$_SERVER['HTTP_REFERER']          リンクの参照元URL(例:http://www.example.com/)
$_SERVER['HTTP_USER_AGENT']       リクエストのUser-Agent:ヘッダの値
(例:Mozilla/5.0...)
$_SERVER['HTTPS']                 HTTPSを利用しているか否か(例:on)
$_SERVER['PHP_AUTH_DIGEST']       ダイジェスト認証時のAuthorization:ヘッダの値
$_SERVER['PHP_AUTH_USER']         HTTP認証時のユーザ名
$_SERVER['PHP_AUTH_PW']           HTTP認証時のパスワード
$_SERVER['AUTH_TYPE']             HTTP認証時の認証形式
```

◇ 変数展開

文字列の中で、変数の中身を取り出すことを『変数展開』と呼ぶ。

※Paizaで検証済み。

- **シングルクオーテーションによる変数展開**

シングルクオーテーションの中身は全て文字列として認識され、変数は展開されない。

```
$fruit = "リンゴ";

echo 'これは$fruitです。';

// 出力結果
これは、$fruitです。
```

- **ダブルクオーテーションによる変数展開**

変数の前後に半角スペースを置いた場合にのみ、変数は展開される。（※半角スペースがないとエラーになる）

```
$fruit = "リンゴ";

echo "これは $fruit です。";

// 出力結果
これは リンゴ です。
```

- **ダブルクオーテーションと波括弧による変数展開**

波括弧を用いると、明示的に変数として扱うことができる。これによって、変数の前後に半角スペースを置かなくとも、変数は展開される。

```
$fruit = "リンゴ";

echo "これは{$fruit}です。";

// 出力結果
これは、リンゴです。
```

◇ 参照渡しと値渡し

- **参照渡し**

「参照渡し」とは、変数に代入した値の参照先（メモリアドレス）を渡すこと。

```
$value = 1;
$result = &$value; // 値の入れ物を参照先として代入
```

【実装例】 `$b` には、`$a` の参照によって10が格納される。

```
$a = 2;
$b = &$a;  // 変数aを&をつけて代入
$a = 10;    // 変数aの値を変更
echo $b;

# 結果
10
```

- **値渡し**

「値渡し」とは、変数に代入した値のコピーを渡すこと。

```
$value = 1;
$result = $value; // 1をコピーして代入
```

【実装例】 `$b` には、`$a` の一行目の格納によって2が格納される。

```
$a = 2;
$b = $a;  // 変数aを代入
$a = 10;  // 変数aの値を変更
echo $b;

# 結果
2
```