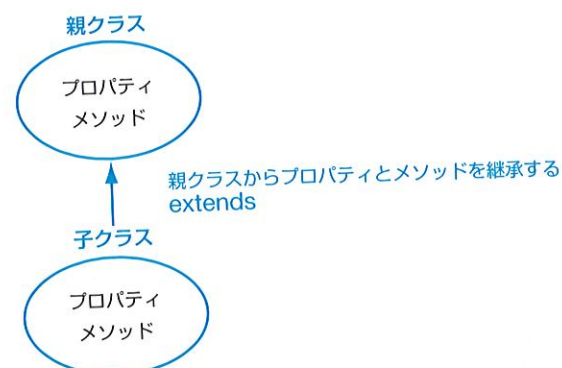


クラスの継承 P.229

OOPではプログラムコードの機能を改変、拡張したいとき「継承」を使います。継承こそがOOPの醍醐味と言えるでしょう。継承ではAクラスに追加したい機能があるとき、Aクラスのコードを書き替えずに、Aクラスを継承したBクラスを作ります。そして、Aクラスに追加したい機能をBクラスに実装します。そうするとBクラスは自身で追加したコードに加えて、Aクラスから継承した機能を兼ね備えたクラスになります。



これは親が子に、師匠が弟子に技術を継承することにも似ています。子は親の記述を授かりますが、さらに自身のアイデアを追加したり、継承した技術をアレンジしたりしてオリジナルのスタイルを確立します。これが継承という手法です。

プログラム言語によって「スーパークラスとサブクラス」、「親クラスと子クラス」、「基底クラスと派生クラス」のように継承関係を違う用語で表現します。PHPの場合はparentキーワードで継承される側のクラスを指し示すので、「親クラスと子クラス」という表現がぴったりでしょう。

extends キーワード

PHPでは継承をextendsキーワードを使って記述します。次の書式で示すように、子クラスが親クラスを指定します。親クラスが自分の子クラスを指定することはできません。したがって図でも「子クラス→親クラス」のように矢印を親クラスに向けて描きます。

書式 クラスの継承

```
class 子クラス extends 親クラス {
}
```

Cookクラスを継承したFrenchCookクラスを定義するならば、次のようなコードになります。

php Cookクラスを継承したFrenchCookクラスを定義する

```
01: class FrenchCook extends Cook {
02:     // FrenchCookで拡張する内容
03: }
```

トレイト P.236

PHPにはトレイト (trait) というコードのインクルード (読み込み) に似た仕組みがあります。トレイトでプロパティやメソッドを定義しておく、クラス定義の最初でuseキーワードでトレイトを指定するだけで、そのトレイトのコードを自分のクラスで定義してあるかのように利用できます。複数のトレイトを採用したり、トレイトを組み合わせる新しいトレイトを作ることできます。トレイトの考え方はシンプルですが、利用する際には名前の衝突などに気を配る必要があります。

書式 トレイトの定義

```
trait トレイト名 {
    // トレイトのプロパティ
    // トレイトのメソッド
}
```

書式 トレイトを利用するクラス

```
class クラス名 {
    use トレイト名;
    // クラスのコード
}
```

——— トレイトで定義してあるプロパティやメソッドを自分のクラスのコードのように利用できるようになります

インターフェース P.242

インターフェースは規格のようなものです。クラスが採用しているインターフェースを見れば、そのクラスで確実に実行できるメソッドと呼び出し方がわかります。Webサービスなどで公開されているAPI (Application Programming Interface) がありますが、ここで使われているインターフェースという言葉と同じ意味合いで理解できます。インターフェースはinterfaceキーワードを付けて宣言して定義し、インターフェースを採用するクラスではimplementsキーワードで指定します。

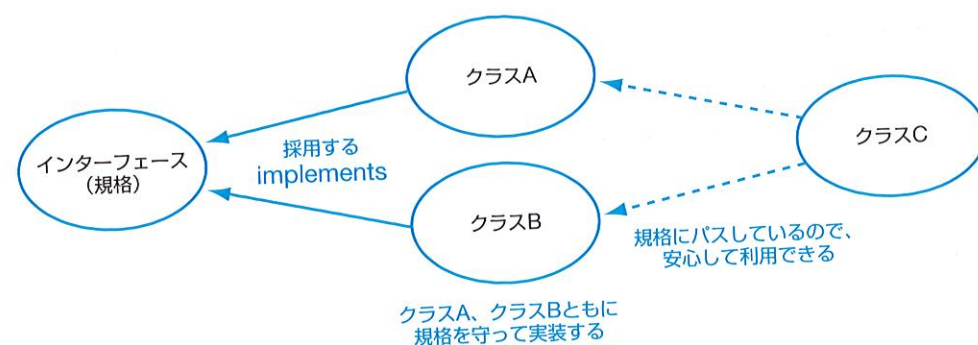
書式 インターフェースの定義

```
interface インターフェース名 {
    function 関数名 ();
}
```

書式 インターフェースを採用するクラス

```
class クラス名 implements インターフェース名 {
    // クラスのコード
}
```

——— このインターフェースを採用します



抽象クラスと抽象メソッド P.247

メソッド宣言のみを行って処理を実装しない特殊なメソッド定義があります。abstract キーワードを付けてメソッド宣言を行うことから抽象メソッドと呼びます。そして、抽象メソッドが1つでもあるクラスには abstract キーワードを付ける必要があり、抽象クラスと呼びます。

抽象クラスのインスタンスを作ることはできず、必ず継承して利用します。そして、抽象メソッドの機能を子クラスでオーバーライド（上書き）して実装します。他の言語と違い、PHPの抽象メソッドには初期機能を実装できません。

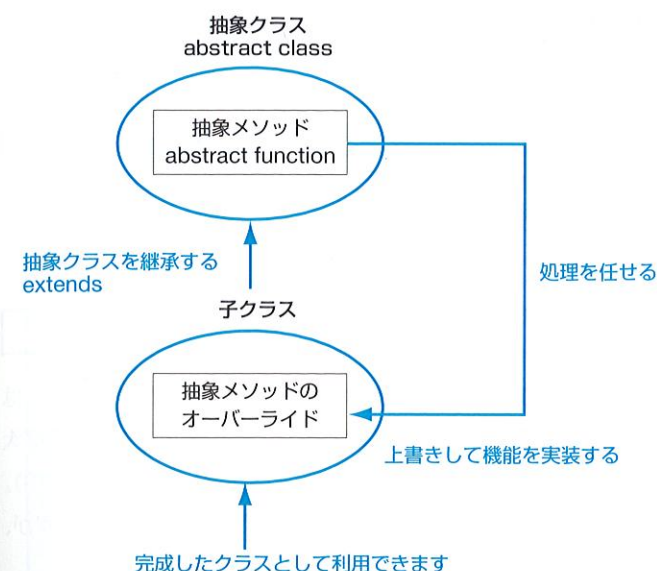
抽象クラスはインターフェースと似た側面がありますが、抽象メソッドだけでなく通常のメソッドを実装できることから、クラスとしての機能をもつことができます。クラス内のメソッドから抽象メソッドを実行することで、実際の処理は子クラスに任せる設計（デリゲート delegate）が可能になります。

書式 抽象クラス

```
abstract class 抽象クラス名 {
    abstract function 抽象メソッド名(); —— メソッド名を宣言するだけで、機能は定義しません
}
```

書式 抽象メソッドを実装する

```
class クラス名 extends 抽象クラス名 {
    function 抽象メソッド名() {
        // メソッドをオーバーライドして機能を定義する
    }
}
```



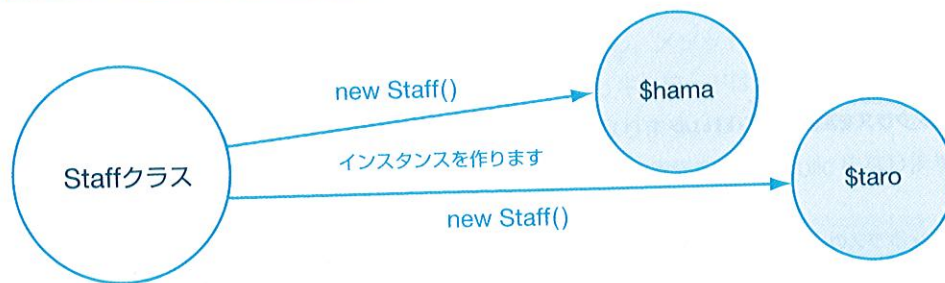
インスタンスを作る

クラスのインスタンスはnew 演算子で作ります。先のStaffクラスのインスタンス\$hanaと\$tarōを作るコードは次のようになります。「new Staff」のようにカッコを付けなくても構いません。

php Staff クラスのインスタンスを作る

«sample» class_Staff.php

```
23: <?php
24: // Staff クラスのインスタンスを作る
25: $hana = new Staff();
26: $tarō = new Staff();
```



インスタンスプロパティのアクセス

Staffクラスには\$name プロパティと\$age プロパティがあります。このプロパティには初期値が設定されていないので、値を設定したいと思います。インスタンスの値は、それぞれのインスタンスに-> 演算子を使ってアクセスします。

書式 インスタンスプロパティにアクセスする

\$ インスタンス -> プロパティ名

次のコードはインスタンス\$hanaと\$tarōのそれぞれの\$name プロパティと\$age プロパティに値を設定している部分です。このとき、「\$hana->name」のようにプロパティ名のnameには\$を付けないので注意してください。

print_r()で確認すると、2個のインスタンスが作られて、それぞれのプロパティに値が設定されていることがわかります。

php インスタンスプロパティに値を設定する

«sample» class_Staff.php

```
27: // プロパティの値を設定する
28: $hana->name = "花";
29: $hana->age = 21;
30: $tarō->name = "太郎";
31: $tarō->age = 35;
32: // インスタンスを確認する
33: print_r($hana);
34: print_r($tarō);
```

出力

```
Staff Object
(
    [name] => 花
    [age] => 21
)
Staff Object
(
    [name] => 太郎
    [age] => 35
)
```



NOTE

\$hana->\$name

「\$hana->\$name」は「\$hana->{ \$name}」と解釈され、同名の変数\$nameに入っている値がプロパティ名として適用されます。

インスタンスメソッドの実行

インスタンスメソッドを実行する場合も同じように-> 演算子を使います。

書式 インスタンスメソッドを実行する

\$ インスタンス -> メソッド ()

\$hanaと\$tarōに対して、hello()を実行するコードは次のとおりです。Staffクラスで定義してあるhello()が実行されて、「こんにちは！」のように出力されます。

出力

250 円になりました。
1250 円になりました。

アクセス修飾子

クラスメンバーのアクセス権は、public、protected、private の3種類のアクセス修飾子で設定します。適切な使い分けには OOP に対する中級者レベルの理解が必要になる場面があります。

修飾子	アクセス権
public	どこからでもアクセスが可能
protected	定義したクラスと子クラスからアクセス可能
private	定義したクラス内のみでアクセスが可能

リードオンリーまたはライトオンリーのプロパティを作りたいといった場合に、protected や private のアクセス修飾子を利用してプロパティの読み書きを禁止に設定し、public なメソッドを介してアクセスできるようにするといった手法を使います。

Section 7-3

クラスの継承

この節ではクラス継承の定義とその使い方を具体的に示します。クラス継承では、親クラスの機能をそのまま利用するだけでなく、上書きして変更することもできます。このオーバーライドと呼ばれる機能を積極的に使うために、親クラスのメソッドを直接指し示すことができたり、逆にオーバーライドを禁止したりすることもできます。

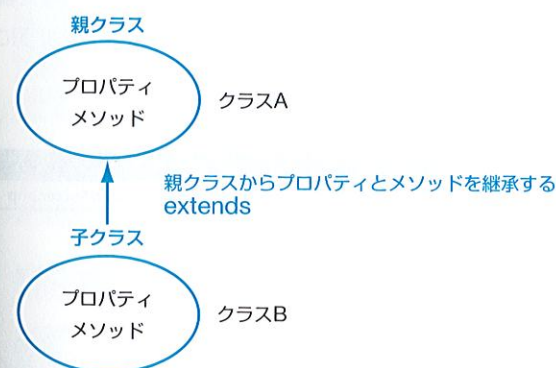
クラスを継承する

クラスの継承とは、既存のクラスを拡張するように自身のクラスを定義する方法です。クラス A をもとにクラス B を作りたいとき、クラス A を継承して追加変更したい機能だけをクラス B で定義します。ベースになるクラス A のコードを改変せずに拡張するので、拡張による影響がクラス A には及ばないというメリットがあります。

クラスの継承には extends キーワードを使います。クラス A を継承してクラス B を作る場合、クラス A が親クラス、クラス B が子クラスという関係になります。

書式 クラスの継承

```
class 子クラス extends 親クラス {
}
```



親クラスの Player クラス

では実際にクラス継承を簡単な例で試してみましょう。まず、親クラスとなる Player クラスを用意します。Player クラスには \$name プロパティ、コンストラクタ、マジックメソッドの __toString()、そして who() メソッドが定義してあります。

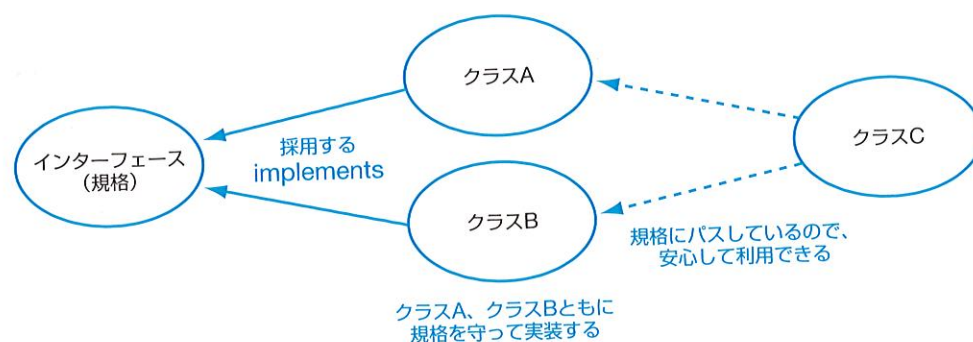
Section 7-5

インターフェース

この節ではインターフェースについて簡単に解説します。インターフェースを効果的に使いこなすには中級者レベルの経験が必要になるかもしれませんが、コードの書き方を理解することは初心者にも難しくありません。

インターフェース

インターフェースはクラスで実装すべきメソッドを規格として定めるものです。たとえば、MyClass クラスが RedBook インターフェースを採用するならば、MyClass クラスは RedBook インターフェースで定められているメソッドを必ず実装しなければなりません。ただ、インターフェースではメソッドの機能については定めていないので、MyClass クラスがメソッドにどんな機能を実装するかについては関知しません。これは、日本の規格に合っている電化製品ならばコンセントに差えますが、どんな機能の製品なのかまでは関知しないのに似ています。



インターフェースを定義する

インターフェースではメソッドと定数を定義できます。メソッドは名前と引数の形式だけを定義し、機能の実装は行いません。アクセス権は public のみが設定可能です。指定を省略すると初期値の public が適用されるので指定する必要はありません。

書式 インターフェースの定義

```
interface インターフェース名 {
    const 定数 = 値;
    function 関数名 ( 引数, 引数, ... );
}
```

ほかのインターフェースを継承したインターフェースも作ることもできます。その場合の書式は次のとおりです。

書式 ほかのインターフェースを継承したインターフェース

```
interface 子インターフェース名 extends 親インターフェース名 {
    const 定数 = 値;
    function 関数名 ( 引数, 引数, ... );
}
```

もっと簡単な例として WorldRule インターフェースを作ってみます。WorldRule インターフェースでは、hello() メソッドの実装だけを指定しています。

php WorldRule インターフェース

«sample» ex_interface/WorldRule.php

```
01: <?php
02: interface WorldRule {
03:     function hello();
04: }
05: // ?>
```

WorldRule インターフェースの規格では、hello() を実装しなければなりません

インターフェースを採用する

インターフェースを採用するクラスでは、implements でインターフェースを指定します。継承と違って、複数のインターフェースを採用できます。

書式 インターフェースを採用するクラス

```
class クラス名 implements インターフェース名, インターフェース名, ... {
    // クラスのコード
}
```

もし、クラスの継承も行う場合は次の書式になります。

書式 インターフェースを採用するクラスに親クラスがある場合

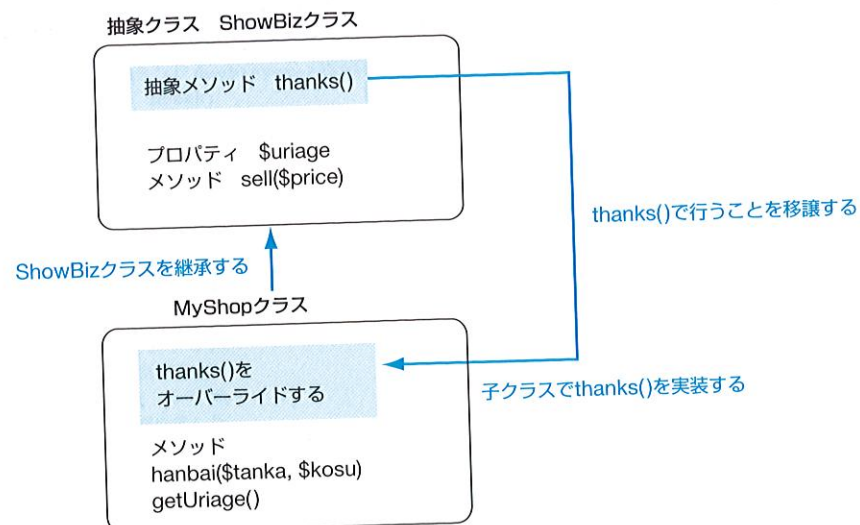
```
class クラス名 extends 親クラス名 implements インターフェース名, インターフェース名, ... {
    // クラスのコード
}
```



```

13: // 子クラスで実装されるメソッドを呼び出す
14: $this->thanks(); // 抽象メソッドの thanks() の機能は、ShowBiz クラスの子クラスで実装します
15: }
16: }
17: // ?>

```



抽象クラスを継承して抽象メソッドを実装する

抽象クラスのインスタンスを直接作ることはできません。抽象クラスは必ず継承して使います。そして、抽象クラスを継承した子クラスでは抽象メソッドを必ずオーバーライドして機能を実装しなければなりません。抽象メソッドにアクセス権が設定されている場合には、子クラスでオーバーライドする場合には同じかそれよりも緩いアクセス権を設定しなければなりません。

書式 抽象クラスを継承して抽象メソッドを実装する

```

class クラス名 extends 抽象クラス名 {
    function 抽象メソッド名() {
        // メソッドをオーバーライドして機能を定義する
    }
    // 子クラスの機能の実装
}

```

ShopBiz 抽象クラスを継承する

次の MyShop クラスは先の ShopBiz 抽象クラスを継承しているクラスです。したがって、ShopBiz クラスで宣言されている抽象メソッドの thanks() をオーバーライドして実装しています。さらに hanbai() を定義し、

その中で親クラスである ShopBiz クラスの sell() を呼び出して使っています。

thanks() は「ありがとうございました。」と表示するだけですが、hanbai() では引数で受け取った単価と個数から金額 \$price を求めて、継承している sell(\$price) を実行しています。getUriage() では、ShopBiz クラスの sell() で加算している uriage プロパティの値を調べて表示します。

php ShopBiz クラスを継承した MyShop クラス

«sample» ex_abstract/MyShop.php

```

01: <?php
02: require_once("ShopBiz.php");
03:
04: class MyShop extends ShopBiz {
05:     // ShopBiz 抽象クラスで指定されているメソッド
06:     public function thanks(){
07:         echo "ありがとうございました。", "\n"; // ShopBiz クラスの抽象メソッド thanks() を実装します
08:     }
09:
10:     // 販売する
11:     public function hanbai($tanka, $kosu){
12:         $price = $tanka * $kosu;
13:         // ShopBiz 抽象クラスから継承しているメソッドを実行
14:         $this->sell($price); // ShowBiz クラスの sell() の中で thanks() が実行されます
15:     }
16:     // 売上合計を調べる
17:     public function getUriage(){
18:         echo "売上合計は、{$this->uriage} 円です。";
19:     }
20: }
21: // ?>

```

MyShop クラスのインスタンスを作って試してみる

それでは MyShop クラスのインスタンス \$myObj を作って、hanbai() と getUriage() を試してみましょう。\$myObj->hanbai(240, 3) を実行すると値段が計算されて sell() に渡され、「720 円です。ありがとうございました。」と表示されます。「ありがとうございました。」は抽象メソッド thanks() をオーバーライドした結果です。\$myObj->getUriage() を実行した結果は「売上合計は、1120 円です。」のように表示されます。

php MyShop クラスのインスタンスを作って試す

«sample» ex_abstract/myShopTest.php

```

01: <?php
02: // MyShop クラスファイルを読み込む
03: require_once("MyShop.php");
04: // MyShop クラスのインスタンスを作って試す
05: $myObj = new MyShop();
06: $myObj->hanbai(240, 3);
07: $myObj->hanbai(400, 1);
08: $myObj->getUriage();
09: ?>

```

出力

720 円です。ありがとうございました。 — ShowBiz クラスの抽象メソッド thanks() に
400 円です。ありがとうございました。 機能が実装されて使われています
売上合計は、1120 円です。