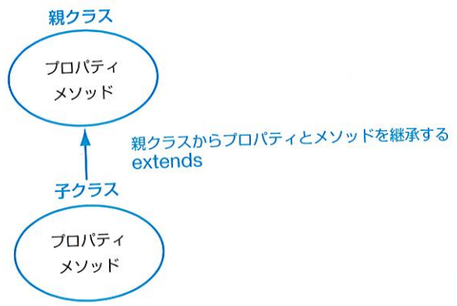
クラスの継承 ☞ P.229

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と、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て新しいトレイトを作ることもできます。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 ;   ——— 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
    // クラスのコード
}
```

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。

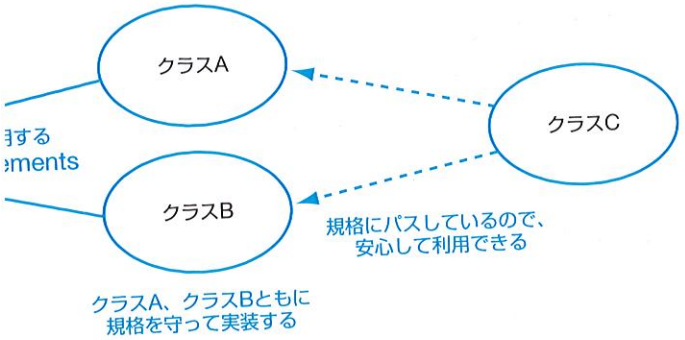
書式 インターフェースの定義

```
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
}
```

インターフェース名 ——— このインターフェースを採用します



×ソッド ☞ P.247

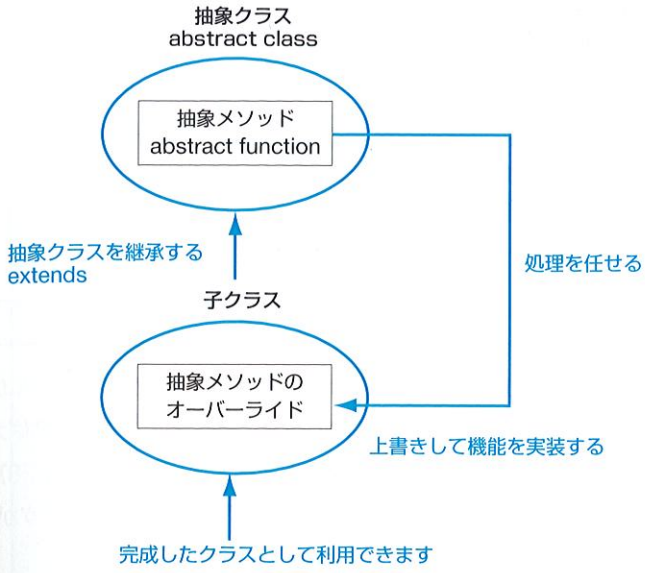
処理を実装しない特殊なメソッド定義があります。abstract キーワードを付けて
う抽象メソッドと呼びます。そして、抽象メソッドが１つでもあるクラスには
必要があり、抽象クラスと呼びます。
を作ることはできず、必ず継承して利用します。そして、抽象メソッドの機能を
上書き）して実装します。他の言語と違い、PHP の抽象メソッドには初期機能を

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
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
スに任せる設計（デリゲート delegate）が可能になります。

```
×クラス名 {
ion 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
```

装する

```
ends 抽象クラス名 {
×ソッド名 () {
 オーバーライドして機能を定義する
```

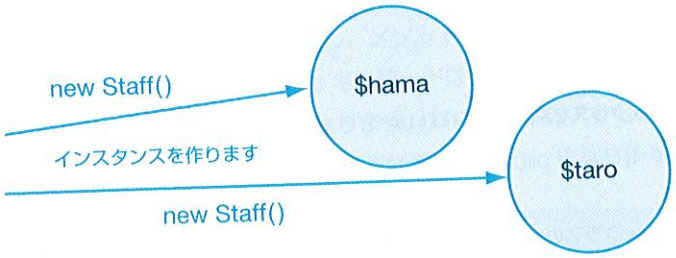



v演算子で作ります。先のStaffクラスのインスタンス$hanaと$taroを作るコー
w Staff」のようにカッコを付けなくても構いません。

スを作る

«sample» class_Staff.php

```
ンスタンスを作る
();
();
```



のアクセス

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
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

ティにアクセスする

コパティ名

ス$hanaと$taroのそれぞれの$nameプロパティと$ageプロパティに値を設定
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

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

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:     $taro->name = "太郎";
31:     $taro->age = 35;
32:     // インスタンスを確認する
33:     print_r($hana);
34:     print_r($taro);
```

$hanaのname、ageプロパティ

$taroのname、ageプロパティ

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()

$hanaと$taro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。

は、public、protected、private の３種類のアクセス修飾子で設定します。適する中級者レベルの理解が必要になる場面があります。

セスが可能
子クラスからアクセス可能
のみでアクセスが可能

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。

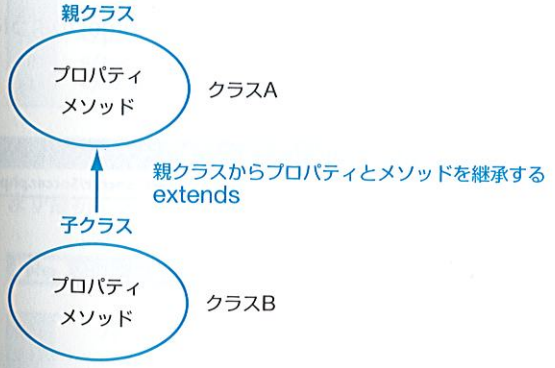
クラスを継承する

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 {
}
```



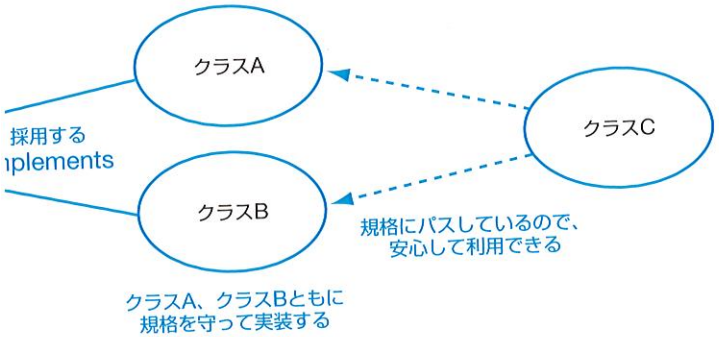
親クラスの Player クラス

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。

ェース

-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
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

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
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
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
ss 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、
製品ならばコンセントに差せますが、どんな機能の製品なのかまでは関知しない



義する

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
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
ません。

の定義

```
フェース名 {
直 ;
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 exitends 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。

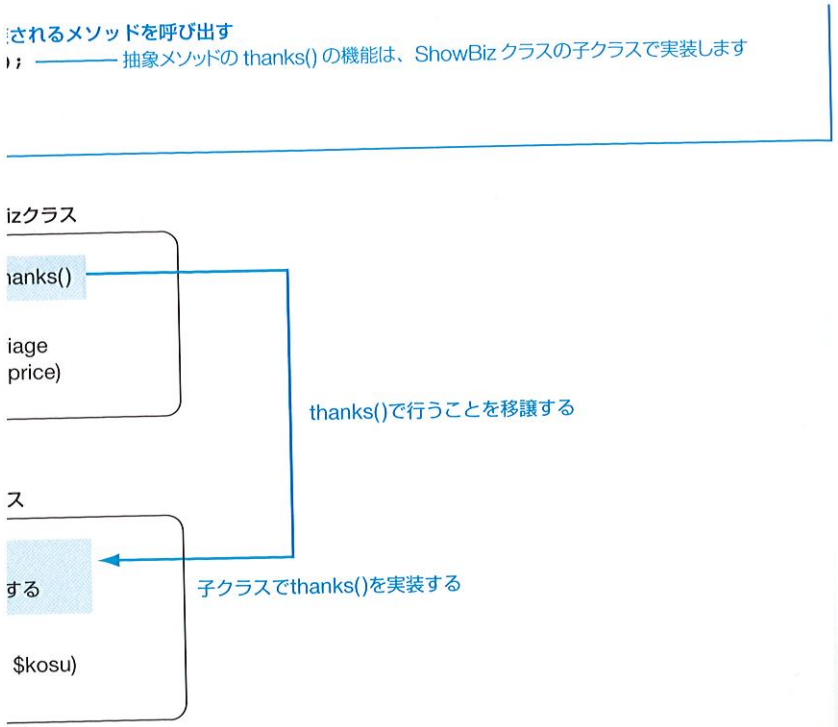
書式 インターフェースを採用するクラス

```
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
}
```



して抽象メソッドを実装する

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
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。
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
なければなりません。

て抽象メソッドを実装する

```
nds 抽象クラス名 {
 ソッド名 () {
オーバーライドして機能を定義する

の実装
```

継承する

この 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

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

```
<?php
require_once("ShopBiz.php");

class MyShop extends ShopBiz {
  // ShopBiz 抽象クラスで指定されているメソッド
  public function thanks(){
    echo "ありがとうございました。", "\n";
  }

  // 販売する
  public function hanbai($tanka, $kosu){
    $price = $tanka * $kosu;
    // ShopBiz 抽象クラスから継承しているメソッドを実行
    $this->sell($price);
  }
  // 売上合計を調べる
  public function getUriage(){
    echo "売上合計は、{$this->uriage} 円です。";
  }
}
// ?>
```

ShopBiz クラスの抽象メソッド thanks() を実装します

ShowBiz クラスの sell() の中で thanks() が実行されます

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

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

出力

```
720 円です。ありがとうございました。
400 円です。ありがとうございました。
売上合計は、1120 円です。
```

ShowBiz クラスの抽象メソッド thanks() に機能が実装されて使われています