Introduction to Object Oriented Programming

Authors:

Ed Lecky-Thompson
Steven D. Nowicki
Thomas Myer

Interpreter:

Masoud Amjadi

مقدمه ای بر برنامه نویسی

شئ گرا

نویسندگان:

Ed Lecky-Thompson
Steven D. Nowicki
Thomas Myer

مترجم:

مسعود امجدی

تیر- 1390©

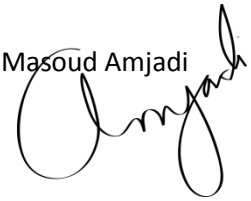
فهرست

Introduction to Object-Oriented Programming

با کلیک بر روی عناوین به صفحه ی مورد نظر هدایت می شوید.

To IRANPHP's members, for being world's best forum.

Masoud Amjadi



کلمات مخفف به کار رفته در این مقاله:

OO = Object Oriented	شی گرایی
OOP = Object Oriented Programmin	برنامه نویسی شی گرا
PRP = Property	مشخصه (پروپرتی)

توسعه ی نرم افزاری شی گرا می تواند موضوعی گیج کننده برای توسعه دهندگانی باشد که قبلا از روش رویه ای (Procedural) برای برنامه نویسی استفاده می کردند، اما لازم نیست این گونه باشد. در این مقاله شما برخی از تئوری های پایه ی OO را کاوش کرده و روش های چند برنامگی را خواهید آموخت. همچنین شما می آموزید که چرا باید از تکنیک های OO استفاده کنید و چگونه این تکنیک ها سرعت توسعه ی برنامه های پیچیده ی شما را بسیار بهبود می بخشند و خواهید دید که تغییر دادن کدهایی که با این روش نوشته شده اند چقدر آسان است.

برنامه نویسی شی گرا چیست؟

برنامه نویسی شی گرا (OOP) نیاز دارد تا شما روش فکر کردن در مورد نحوه ی ساختن برنامه هایتان را تغییر دهید. اشیا شما را قادر می سازند تا کارها، روند ها و ایده های دنیای واقعی را که برنامه تان را به خاطر آنها نوشته اید خیلی خوب توسط کدها مدل سازی کنید. به جای اینکه برنامه تان را به صورت یک رشته کنترلی ببینید که داده ها را از یک تابع به دیگری منتقل می کند، OOP به شما کمک میکند تا برنامه تان را به عنوان مجموعه ای از اشیای هماهنگ مدل سازی کنید که هر کدام از آنها به صورت مجزا از دیگران کارهای خاصی را انجام می دهد.

به عنوان مثال، زمانی که یک خانه ساخته می شود، لوله کش ها با لوله کار میکنند و سیم کش ها با سیم. لوله کش ها نیازی ندارند بدانند که آیا جریان مدار های اتاق خواب 10 آمپر است یا 20 آمپر؟ آنها تنها باید نگران کارهای خود باشند. پیمانکار هم مطمئن است که هر کسی در حال انجام کارهای خود است و نیازی ندارد در جریان ریزه کاری ها قرار بگیرد. در OOP نیز هر شی جزئیات پیاده سازی خود را از دیگر اشیا مخفی نگه می دارد. چگونگی انجام کارها توسط یک شی، هیچ ربطی به دیگر اعضای سیستم ندارد. تمام آنچه که اهمیت دارد سرویسی است که آن شی می تواند ارائه دهد.

کلاس ها، اشیا، و روش هایی که شما این ایده ها را در برنامه هایتان پیاده سازی می کنید، اساسی ترین ایده های پشت سر OOP هستند. این کار در واقع کاملا متضاد برنامه نویسی رویه ای است که در آن از توابع و ساختار های کلی (Global) استفاده می شد. همان طور که خواهید دید یک دیدگاه OO مزایای بسیار مهمی نسبت به برنامه نویسی رویه ای دارد و با پیاده سازی جدید OO که برای اولین بار در php5 انجام گرفت و اکنون در php6 نیز ادامه یافته، برخی از عملکرد های بزرگ هم بهبود پیدا کرده اند.

مزایای برنامه نویسی شی گرا

یکی از مزیت های اساسی OOP، راحتی تبدیل احتیاجات تجاری به ماژول های کد می باشد. از آنجایی که رویکرد OOP شما را قادر می سازد تا برنامه هایتان را بر اساس اشیای دنیای واقعی مدل سازی کنید، شما می توانید یک رابطه ی مستقیم بین مردم، اشیا، مفاهیم و کلاس های هم ارز تعریف کنید. این کلاس ها همان ویژگی ها و رفتار اشیای واقعی را دارند که آنها را نشان می دهند؛ این امر به شما کمک می کند تا خیلی سریع و راحت کد هایی که باید نوشته شوند و همچنین نحوه ی ارتباط اجزای مختلف برنامه را معین کنید.

مزیت دوم OOP توانایی استفاده ی دوباره از کد است. شما دائما به داده های یکسانی در بخش های مختلف برنامه نیاز خواهید داشت. برای مثال، برنامه ای را در نظر بگیرید که یک بیمارستان را قادر می سازد تا اسناد بیمارانش را مدیریت کند. این برنامه مسلما به کلاسی با نام person نیاز خواهد داشت. این کلاس شامل تمامی افرادی می باشد که در امر مراقبت از بیمار نقش دارند، شامل : خود بیمار، دکتر ها، پرستاران، مدیران بیمارستان، مسئولین بیمه و ... . در هر مرحله از مراقبت، لازم است تا یادداشتی در مورد فردی که کاری را برای بیمار انجام می دهد، قرار گیرد و همچنین بررسی شود که آیا آن فرد اجازه انجام آن کار را دارد یا نه؟ با تعریف یک کلاس عمومی با نام person که شامل مشخصات و کار های معمول تمامی این افراد می باشد،

شما بخش بزرگی از کد را قابل استفاده مجدد می کنید و می توانید از آن چندین بار استفاده کنید، در حالی که این کار همیشه در برنامه نویسی رویه ای امکان پذیر نیست.

در مورد برنامه های دیگر چه؟ فکر می کنید در چند برنامه می توان از این کلاس person استفاده کرد؟ شاید تعداد کمی. اما از یک کلاس person که اصولی نوشته شده باشد با اندکی تغییر یا شاید هم بدون هیچ تغییری، به راحتی میتوان در دیگر پروژه ها استفاده کرد. این یکی از بزرگترین مزیت های OOP می باشد؛ " استفاده چند باره از کد در برنامه ی نوشته شده و همچنین دیگر برنامه ها."

مزیت دیگر OOP از پیمانه ای بودن کلاس ها حاصل می گردد. اگر شما اشکالی (bug) را در کلاس person پیدا کنید و یا بخواهید یک تابع به آن بیفزایید یا یکی از توابعش را تغییر دهید؛ تنها باید به یک جا مراجعه کنید. تمامی کد های کلاس در یک فایل ذخیره می شوند و هر تغییری در آنها فورا در کل برنامه نمایان می شود. این مزیت به طور فوق العاده ای جستجو به دنبال اشکالات و یا ایجاد تغییرات در برنامه را آسان می کند.

یک مثال واقعی

اگر چه مزایای پیمانه ای ممکن است در یک برنامه ی کوچک ناچیز به نظر برسد، اما در معماری یک نرم افزار پیچیده این مزایا خیلی بزرگ و مهم خواهند بود. یکی از نویسندگان این کتاب بر روی پروژه ای شامل بیش از 200,000 کد رویه ای php کار می کند و به راحتی می توان گفت 65 درصد از زمان رفع اشکال برای یافتن جای توابع در کد و همچنین فهمیدن اینکه این داده را کدام تابع تولید کرده است، سپری می شود. پیاده سازی دوباره همان نرم افزار با معماری OO باعث شد تا کد ها به طور چشمگیری کاهش یابند. پیاده سازی با معماری OO نه تنها باعث کاهش کد ها شد، بلکه تعداد اشکالات برنامه و زمانی که برای رفع اشکال مورد نیاز بود نیز بسیار کاهش یافت.

پروژه های بزرگ اغلب یک تیم چند نفره برای توسعه ی نرم افزار دارند، که معمولا ترکیبی از برنامه نویسان با سطوح مهارتی مختلف هستند. در اینجا نیز یک رویکرد OO مزیت های مهمی نسبت به برنامه نویسی رویه ای دارد. اشیا جزئیات پیاده سازی خود را از کاربرانشان مخفی نگه می دارند. بنابرین بدون نیاز به فهم ساختار های پیچیده ی اطلاعاتی و تمامی خصوصیات منطق تجاری، اعضای تازه کار تیم می توانند از اشیایی که دیگر اعضای با تجربه ی تیم درست کرده اند استفاده کنند. خود اشیا مسئول تغییراتی که بر روی داده ها صورت می گیرد و وضعیت سیستم هستند.

هنگامی که هنوز برنامه بزرگ اشاره شده در بالا با استفاده از کدهای رویه ای نوشته شده بود، اغلب دو ماه طول می کشید تا اعضای جدیدی که می خواستند به تیم بپیوندند با برنامه آشنا شوند و بتوانند مفید واقع شوند. بعد از اینکه برنامه با اشیا پیاده سازی شد، اعضای جدید معمولا به بیش از دو روز برای فهم برنامه نیاز نداشتند. آنها می توانستند حتی پیچیده ترین اشیا را به سرعت استفاده کنند، چون نیازی به دانستن جزئیات توابع و پیاده سازی آنها نداشتند.

اکنون شما دلیل مناسبی برای استفاده از الگوی OO به عنوان روش برنامه نویسیتان دارید.

جنبه های برنامه نویسی شی گرا

این بخش به معرفی اصول اولیه ی برنامه نویسی شی گرا و چگونگی فعل و انفعال آنها می پردازد. در این بخش به موضوعات زیر می پردازیم:

- **کلاس ها:** طرح اولیه یک شی و کدی که مشخصات و متد های آن را تعریف می کند.
- **اشیا:** نمونه هایی از یک کلاس که شامل تمامی داده های داخلی و اطلاعاتی است که برنامه ی شما به آنها نیاز دارد.

- **وراثت:** توانایی تعریف یک کلاس به عنوان زیرکلاسی از یک کلاس دیگر (مثل اینکه بگوییم مربع خود نوعی مستطیل است.)
- **چند ریختی (Polymorphism):** اجازه ی تعریف یک کلاس به عنوان عضوی از چند طبقه بندی کلاسی (برای مثال: ماشین هم یک "شی موتوردار است" و هم یک "شی چرخدار")
- **واسط ها:** روشی برای مشخص کردن اینکه یک شی می تواند یک کار را انجام دهد، بدون اینکه دقیقا نحوه ی انجام آن کار را مشخص کنیم. (برای مثال، یک سگ و یک انسان اشیایی هستند که می توانند راه بروند، اما آنها این کار را بسیار متفاوت انجام می دهند).
- **کپسول سازی:** توانایی یک شی برای جلوگیری از دسترسی دیگران به اطلاعات داخلی اش.

اگر برخی از این جنبه ها مشکل به نظر می رسند، نگران نباشید. روشی را که دنبال می کنید همه چیز را روشن خواهد کرد. دانش جدیدی که یافته اید ممکن است به طور کامل رویکرد توسعه ی نرم افزاری شما را تغییر دهد.

کلاس ها

در دنیای واقعی اشیا مشخصات و رفتار خاص خود را دارند. یک ماشین رنگ، ارتفاع، یک سازنده و یک باک با حجم مشخص دارد. این ها مشخصات یک ماشین هستند. یک ماشین می تواند شتاب بگیرد، بایستد، راهنما بزند و یا با بوق هشدار دهد. این ها رفتاری هستند که یک ماشین از خود نشان می دهد. این مشخصات و رفتار ها در مورد تمامی ماشین ها صدق می کنند. اگر چه دو ماشینی که در یک جا کنار هم پارک ممکن است رنگ های متفاوتی داشته باشند، ولی همه ی ماشین ها رنگ دارند. با استفاده از ساختاری که به آن class می گویند، OOP شما را قادر می سازد تا ماشینی را با مشخصات بالا پیاده سازی کنید. یک کلاس کد واحدی است که

مشخصات و رفتار تمامی اعضای یک مجموعه را تعریف می کند. کلاسی با عنوان car می تواند مشخصات و متد های معمول تمامی ماشین ها را تعریف کند.

در اصطلاحات OOP، به مشخصات یک کلاس Properties یا همان خواص می گویند(در ادامه Property را با نماد PRP نشان خواهیم داد). PRP ها هر کدام یک اسم و یک مقدار دارند. برخی از آنها اجازه می دهند مقادیرشان تغییر کند و برخی اجازه نمی دهند. برای مثال در کلاس car ممکن شما PRP های car و weight را داشته باشید. اگرچه رنگ ماشین را می توان با دوباره رنگ کردن تغییر داد، اما وزن ماشین مقدار ثابتی خواهد داشت(فرض کنید که به ماشین بار یا مسافری اضافه نگردد).

بعضی از PRP ها وضعیت و حالت(state) شی را نشان می دهند. state ها خواصی از شی را مشخص می کنند که با توجه به وقوع اتفاقات خاصی مقدارشان تغییر می کند و به تنهایی نمی توانند مقدار خود را تغییر دهند. در برنامه ای که عملکرد یک ماشین را شبیه سازی می کند، ممکن است کلاس car یک PRP با نام velocity (سرعت) داشته باشد. سرعت ماشین متغیری نیست که بتواند به تنهایی تغییر کند، بلکه با توجه به مقدار سوختی که به موتور می رسد، عملکرد موتور و زمینی که ماشین در روی آن در حال حرکت تغییر می کند.

رفتار های کلاس را متد (Method) می گویند. متد ها در OOP مشابه همان توابعی هستند که در برنامه نویسی رویه ای سنتی وجود داشت. همانند توابع، متدها نیز می توانند هر چندتا پارامتر از هر نوعی را که خواستند بگیرند. بعضی از متدها روی داده هایی کار می کنند که از بیرون به آنها به عنوان پارامتر ارسال می گردد. آنها همچنین می توانند بر روی PRP های اشیای خود نیز کار کنند و یا از آنها برای فعال کردن یک اکشن خاص از متد استفاده کنند.

اشیا

برای شروع می توانید کلاس را همانند یک نقشه ی ساختمانی در نظر بگیرید که قرار است از آن اشیایی ساخته شود. همان طور که می توان چندین خانه از یک نقشه درست کرد، شما

نیز می توانید هر چندتا شی که خواستید از یک کلاس بسازید. می دانیم که نقشه ساختمان جزئیات رنگ دیوارها یا کف زمین را مشخص نمی کند؛ بلکه صرفا وجود آنها را تعیین می کند. کلاس ها بسیار شبیه به این کار می کنند. کلاس مشخصات و رفتار اشیا را مشخص می کند اما ضرورتی ندارد که به آنها مقدار نیز بدهد. یک شی یا object موجودیتی است که با توجه به نقشه ی کلاس ساخته شده است. شما می توانید خانه را کلاس و خانه ی خود را یک شی در نظر بگیرید.

با استفاده از نقشه ای که در دست دارید و کمی مصالح ساختمانی شما می توانید یک خانه بسازید. در OOP، به فرآیند ایجاد شی با استفاده از کلاس Instantiation می گویند. برای تولید یک شی به دو چیز نیاز دارید:

- حافظه ای که شی ساخته شده در آن قرار بگیرد. این کار را php به طور اتوماتیک برای شما انجام می دهد.
- داده هایی که باید به عنوان مقادیر PRP ها در شی جایگزین گردند. این داده ها می توانند از یک دیتابیس خوانده شوند و یا از اشیای دیگر گرفته شوند و یا از دیگر منابع به دست آیند.

یک کلاس هرگز نمی تواند مقادیر PRP ها و یا state را داشته باشد. این مقادیر تنها می توانند در اشیا قرار بگیرند. شما باید اول با استفاده از نقشه خانه را بسازید، سپس به دیوار هایش رنگ بزنید یا کارهای دیگر را انجام دهید. به همین ترتیب ابتدا شما باید با استفاده از کلاس شی را بسازید سپس بتوانید از PRP هایش استفاده کنید یا متد هایش را فراخوانی کنید. این که کجا از واژه ی class استفاده کنیم و کجا از object، مساله ایست که گاهی باعث سردرگمی تازه کاران در OOP می شود.

بعد از اینکه یک شی ایجاد شد می توان از برای پیاده سازی احتیاجات تجاری برنامه استفاده کرد. اجازه بدهید نحوه ی انجام این کارها را در php بررسی کنیم.

ایجاد یک کلاس

از یک مثال ساده شروع می کنیم. کد زیر را در فایلی با عنوان class.Demo.php ذخیره کنید:

```
<?php
	class Demo {
	}
?>
```

اکنون شما کلاس Demo را ایجاد کرده اید. اگرچه هنوز زیاد هیجان انگیز نشده است، اما کد بالا نحوه ی تعریف یک کلاس جدید در php را نشان می دهد. از کلید واژه ی class استفاده کنید تا php بفهمد که شما تصمیم دارید یک کلاس جدید ایجاد کنید. بعد از آن اسم کلاستان را بنویسید و شروع و پایان کد کلاس را با آکولاد های باز و بسته مشخص کنید.

بسیار مهم است که قراردادی برای تعریف فایل های سورس کد خود داشته باشید. یک روش خوب این است که هر کلاس را در فایلی به اسم خودش ذخیره کنید به صورت زیر:

class.[ClassName].php

شما با استفاده از کد زیر میتوانید یک شی از کلاس Demo بسازید:

```
<?php
	require_once('class.Demo.php');
	$objDemo = new Demo();
?>
```

برای ایجاد یک شی ابتدا با استفاده از require_once() به php بفهمانید که فایل کلاس شما در کجا قرار دارد، سپس با فراخوانی عملگر new و نام کلاس بعد از آن، شی جدید را ایجاد کنید. مقداری را که این جمله بر می گرداند به یک متغیر جدید که در این مثال $objDemo می باشد، نسبت دهید. اکنون شما می توانید متد های شی $objDemo را فراخوانی کنید یا به PRP های آن در صورت وجود مقدار بدهید.

اگرچه کلاسی که شما تعریف کرده اید فعلا هیچ کار خاصی انجام نمی دهد، اما به درستی تعریف شده است.

افزودن یک متد

کلاس Demo اگر نتواند کاری انجام دهد به هیچ دردی نمی خورد، پس بیایید ببینیم چگونه می توانیم یک متد به آن اضافه کنیم. به یاد بیاورید که متد های کلاس همان توابع یا function ها هستند. با نوشتن یک تابع در داخل کلاس شما یک متد به آن اضافه کرده اید. به مثال زیر توجه کنید:

```
<?php
	class Demo {
		function sayHello($name) {
			print "Hello $name!";
		}
	}
?>
```

شی ای که از کلاس خود ساخته اید اکنون قادر است به کسانی که متد sayHello را فراخوانی می کنند، سلام کند. در شی $objDemo می توانید از عملگر -> برای دستیابی به متدها استفاده کنید:

```
<?php
	require_once('class.Demo.php');
	$objDemo = new Demo();
	$objDemo -> sayHello('Masoud');
?>
```

عملگر -> برای دسترسی به تمامی متد ها و PRP های اشیای شما به کار می رود. کسانی که قبلا با مفهوم OOP در دیگر زبان های برنامه نویسی کار کرده اند باید توجه کنند که همیشه باید از عملگر -> برای دستیابی به متد ها و PRP ها استفاده کنند و عملگر نقطه (.) اصلا در گرامر OO تعریف نشده است.

افزودن یک PRP

افزودن یک PRP به کلاس نیز به همان آسانی افزودن متد می باشد. شما به سادگی متغیری را در داخل کلاس تعریف می کنید تا مقدار PRP را نگه دارد. زمانی که در برنامه نویسی رویه ای می خواستید مقداری را ذخیره کنید آنرا به متغیری نسبت می دادید. در OOP هم زمانی که شما می خواهید مقدار یک PRP را ذخیره کنید از متغیر ها استفاده می کنید. این متغیر ها در ابتدای کلاس تعریف می شوند. نام متغیر همان نام PRP است. اگر متغیری با عنوان $color باشد، ما PRP ای با عنوان color خواهیم داشت.

کلاس class.Demo.php را باز کرده و تغییرات مشخص شده زیر را در آن وارد کنید:

```
<?php
	class Demo {
		public $name;
		function sayHello() {
			print "Hello $this -> name!";
		}
	}
?>
```

در کد بالا متغیر $name نماینده ی PRP ای به عنوان name برای کلاس Demo می باشد. برای دستیابی به این PRP نیز شما باید از عملگر -> استفاده کنید.

فایل جدیدی با نام testdemo.php ایجاد کرده و کد زیر را در آن قرار دهید:

```
<?php
	require_once('class.Demo.php');

	$objDemo = new Demo();
	$objDemo -> name = 'Masoud';

	$objAnotherDemo = new Demo();
	$objAnotherDemo -> name = 'Ed';

	$objDemo -> sayHello();
	$objAnotherDemo -> sayHello();
?>
```

فایل را ذخیره کرده و سپس با استفاده از مرورگرتان باز کنید. دو جمله ی " Hello Masoud" و "Hello Ed" در صفحه چاپ خواهند شد.

واژه کلیدی public به کلاس می فهماند که شما می خواهید به این متغیر از خارج کلاس نیز دسترسی داشته باشید. برخی از متغیر های کلاس تنها برای دسترسی خود کلاس هستند و دیگران نباید به آنها دسترسی داشته باشند، برای تعریف این نوع متغیر ها از private یا protected استفاده می کنیم. توجه کنید که در مثال بالا نحوه ی عملکرد متد sayHello تغییر کرد و به جای آنکه یک پارامتر بگیرد، مقدار name را از PRP می گیرد.

شما از متغیر $this استفاده کردید، در نتیجه شی می تواند به اطلاعات خودش دست پیدا کند. برای مثال ممکن است شما چندین شی از یک کلاس داشته باشید، در این صورت چون نمی دانید در ادامه ی اسم متغیر یک شی چه خواهد بود، می توانید از $this برای اشاره به نمونه ی کنونی استفاده کنید.

در مثال قبلی فراخوانی اول متد sayHello مقدار Masoud و فراخوانی دومش مقدار Ed را چاپ می کند. این بدین دلیل است که متغیر $this به هر شی ای اجازه می دهد تا به PRP ها و متد های خودش دسترسی داشته باشد، بدون اینکه لازم باشد اسم متغیری که آن را در برنامه ی خارجی نشان می دهد، بداند. قبلا دیدید که برخی از PRP ها بر روی عملکرد متد های خاصی تاثیر می گذاشتند، همانند مثالی که در آن متد accelerate کلاس car نیاز داشت تا مقدار سوخت باقی مانده را بداند. در داخل متد accelerate باید از کدی مشابه کد زیر برای دستیابی به آن PRP استفاده کرد.

```
$this -> amountofFuel;
```

توجه کنید که هنگام دسترسی به PRP ها باید فقط از یک علامت $ استفاده کنید. در نتیجه کد $obj -> $property غلط بوده و به جای آن باید از $obj -> property استفاده کرد.

علاوه بر متغیر هایی که مقادیر PRP های کلاس را نگه می دارند، ممکن است متغیر های دیگری نیز برای کار های داخلی خود کلاس تعریف شوند. هر دو نوع این متغیر ها به عنوان متغیر های داخلی کلاس شناخته می شوند. برخی از این متغیر ها در قالب PRP ها از خارج کلاس نیز قابل دستیابی هستند. بقیه آنها تنها برای کارهای داخلی خود کلاس هستند. برای مثال اگر کلاس car به هر دلیلی نیاز داشته باشد تا اطلاعاتی را از دیتابیس بخواند، اطلاعات مربوط به اتصال را به عنوان متغیر های داخلی نگه می دارد. چرا که این اطلاعات به عنوان PRP برای ماشین نیستند اما کلاس برای انجام فعالیت های خاصی به آنها نیاز دارد.

حفاظت از دسترسی به متغیر های عضو

همان طور که در مثال قبل دیدید شما می توانید به name (منظور همان PRP می باشد) هر مقدار معتبری از جمله شی، آرایه ای از اعداد، یک فایل و ... را بدهید. توجه کنید که شما بعد از نسبت دادن مقداری به PRP دیگر فرصت بررسی صحت داده های آن و یا بروزرسانی آنها نخواهید داشت.

برای غلبه بر این مشکل همیشه PRP ها را با استفاده از دو متد get [property name] و set [property name] پیاده سازی کنید. این توابع را متدهای دستیابی می نامند که در مثال پایین نحوه ی استفاده از آنها نشان داده شده است. فایل class.Demo.php را مطابق کد زیر تغییر دهید:

```
<?php
	class Demo {
		private $_name;

		public function sayHello() {
			print "Hello" . $this -> name . "!";
		}

		public function getName() {
			return $this -> name;
		}
```

```
            public function setName($name) {
                  if (!is_string($name) || strlen($name)==0) {
                        throw new Exception("Invalid name value!");
                  }
                  $this -> name = $name;
            }
      }
?>
```

در فایل testdemo.php نیز تغییرات زیر را اعمال کنید:

```
<?php
      require_once('class.Demo.php');
      $objDemo = new Demo();
      $objDemo -> setName('Masoud');
      $objDemo -> sayHello();
      $objDemo -> setName(37); // برای این مورد خطا خواهد داد
?>
```

همان طور که دیدید سطح دسترسی name از public به private تغییر پیدا کرد و اسم آن با یک _ شروع شد. پیشنهاد می شود برای نام گزاری اعضای private کلاس از _ در ابتدای آن استفاده کنید. دقت کنید که php هیچ نیازی به این نحوه ی نام گزاری ندارد و این صرفا قراردادی برای نام گزاری است. کلید واژه ی private به کد های خارج از کلاس اجازه نمی دهد تا بتوانند مقادیر این متغیر را تغییر دهند. مقادیر اعضای داخلی private از خارج کلاس قابل دستیابی نیستند و به همین دلیل شما مجبورید از متد های getName() و setName() برای دسترسی به این اطلاعات استفاده کنید، و در عین حال مطمئن خواهید بود که کلاس شما می تواند صحت مقادیر نسبت داده شده را قبل از انتساب بررسی کند.

در این مثال اگر مقدار نامعتبری به name نسبت داده شود یک استثنا برگردانده می شود. همچنین برای دیگر اعضای کلاس سطح دسترسی public افزوده شده است. اگر برای یک عضو کلاس سطح دسترسی مشخص نشود به صورت پیش فرض مقدار آن public در نظر گرفته می شود، اما بهتر است برای تمرین هم که شده تمامی سطوح دسترسی ها را مشخص کنید.

سه سطح دسترسی مختلف برای اعضای کلاس وجود دارد، public، private و protected. اعضای public برای تمامی کد ها قابل دستیابی هستند. اعضای private تنها برای خود کلاس هستند و اعضای protected قابل دستیابی برای خود کلاس و همچنین کلاس هایی که از آن ارث می برند، هستند. (در ادامه ی همین مقاله به مبحث وراثت پرداخته می شود).

با افزودن متد های دستیابی (accessor methods) به تمامی PRP های کلاس، شما امکان بررسی صحت داده ها و بروز رسانی آنها در آینده را بسیار آسان تر می کنید. حتی اگر برنامه ی کنونی شما نیازی به بررسی صحت داده ها و یا بروز رسانی آنها نداشته باشد، باز هم شما باید از متد های دستیابی برای پیاده سازی PRP های خود استفاده کنید. چرا که بعد ها به آن موارد نیاز پیدا خواهید کرد.

مقداردهی اولیه به اشیا

برای اغلب کلاس هایی که خواهید نوشت، شما به انجام برخی تنظیمات اولیه به هنگام ایجاد بعضی از اشیا برای اولین بار، نیاز خواهید داشت. برای مثال ممکن است لازم داشته باشید مقادیری را از دیتابیس بخوانید یا به بعضی از PRP ها مقدار اولیه بدهید. با ایجاد متد خاصی که constructor یا سازنده نام دارد و در php با فراخوانی تابع ()construct_ پیاده سازی می شود، شما می توانید این کارها را انجام دهید. php خودش به طور اتوماتیک به هنگام ساختن یک شی این تابع خاص را فراخوانی می کند. برای مثال شما می توانید کلاس Demo را به صورت زیر بازنویسی کنید:

```
<?php
      class Demo {
            private $name;

            public function _construct($name) {
                  $this -> $name;
            }
            function sayHello() {
                  print "Hello $this -> name!";
```

```
            }
        }
?>
```

در php4، سازنده ها توابعی با اسم خود کلاس بودند. در php5 این روش به یک اسکمای یکپارچه تبدیل شده است. یعنی برای تمامی کلاس ها از یک نوع سازنده و با اسم مشخص استفاده می شود. برای هماهنگی با نسخه های قبلی ابتدا php به دنبال تابعی با نام ()construct_ می گردد، اما اگر چنین تابعی پیدا نکرد به دنبال تابعی هم اسم با کلاس می گردد (در مثال بالا به دنبال تابعی با نام ()public function Demo می گردد). اگر چه این سازگاری با نسخه های قبلی در php6 نیز وجود دارد، اما هیچ تضمینی برای نسخه های بعدی وجود ندارد. بنابرین بهتر است از روش جدید استفاده کنید.

اگر شما کلاسی دارید که به هیچ مقدار دهی اولیه ای نیاز ندارد، لزومی ندارد که از سازنده استفاده کنید. همان طور که در ورژن اول کلاس Demo دیدید،php خودش تمام کارهای لازم برای ایجاد یک شی جدید را انجام می دهد. از تابع سازنده تنها زمانی که به آن نیاز دارید استفاده کنید.

تخریب اشیا

اشیایی را که ایجاد می کنید در سه حالت زیر از بین می روند :

- زمانی که کار با صفحه ی مورد نظر تمام شود.
- زمانی که مقادیر از حوزه ی تعریف خود خارج شوند.
- و یا زمانی که مقدار NULL به آنها داده شود.

در php6 شما می توانید فرآیند تخریب اشیا (object destruction) را مدیریت کرده و اقداماتی را که می خواهید به هنگام تخریب اشیا صورت بگیرد، انجام دهید. برای انجام این کار تابعی بدون پارامتر و با عنوان destruct_ بسازید. اگر این تابع را ایجاد کنید قبل از تخریب شی به طور اتوماتیک فراخوانی می شود.

استفاده از این تابع به شما این امکان را می دهد که هر کاری را که دوست دارید در لحظات آخر قبل از تخریب شی انجام دهید. برای مثال اتصالات به دیتابیس را قطع کرده و یا فایل های گشوده شده را ببندید.

مثال زیر PRP های یک شی را از دیتابیس می خواند و اگر هر کدام از آنها تغییری کرد آن تغییر را به هنگام تخریب آن شی به طور اتوماتیک در دیتابیس ذخیره می کند. برای این کار نیاز به فراخوانی یک متد برای ذخیره داریم. مخرب (destructor) همچنین اتصال دیتابیس را قطع می کند.

در این مثال و دیگر مثال های که در آن به دیتابیس نیاز است از پلتفرم PostgreSQL استفاده شده است. نویسندگان این اثر به طور جدی معتقد هستند که ویژگی های برتر، توانایی حمایت از تراکنش ها و مکانیسم قوی PostgreSQL در ذخیره ی اطلاعات، PostgreSQL را نسبت به دیگر سیستم های مدیریت دیتابیس مثل MySQL و دیگر سیستم های متن باز، برای انجام کار های بزرگ، برتری می بخشد. اگر شما پلتفرم PostgreSQL را بر روی سیستم خود ندارید و یا به هر دلیلی دیگر سیستم ها را می پسندید، به راحتی می توانید کد های دیتابیس را به کد های پلتفرم خود تغییر دهید.

جدولی به اسم widget به صورت زیر درست کنید:

```
CREATE TABLE "widget" {
      "widgetid" SERIAL PRIMARY KEY NOT NULL,
      "name" varchar(255) NOT NULL,
      "description" text
};
```

داده هایی را وارد آن کنید:

```
INSERT INTO "widget" ("name" , "description")
VALUES ('Foo' , 'This is a footacular widget!');
```

کلاسی با عنوان class.Widget.php درست کرده و اطلاعات زیر را در آن وارد کنید:

```
<?php
class Widget {
      private $id;
```

```
    private $name;
    private $description;
    private $hDB;
    private $needsUpdating = false;
    public function _construct($widgetID) {
    //The widgetID parameter is the primary key of a
    //record in the database containing the information
    //for this object
    //Create a connection handle and store it in a private member variable
    //This code assumes the DB is called "parts"

       $this-> hDB = pg_connect('dbname=parts user=postgres');
       if(! is_resource($this-> hDB)) {
          throw new Exception('Unable to connect to the database.');
       }
       $sql = "SELECT \"name", \"description\" FROM widget WHERE widgetid =
              $widgetID";
       $rs = pg_query($this -> hDB, $sql);
       if(! is_resource($rs)) {
          throw new Exception("An error occurred selecting from the
          database.");
       }
       if(! pg_num_rows($rs)) {
          throw new Exception('The specified widget does not exist!');
       }
       $data = pg_fetch_array($rs);
       $this-> id = $widgetID;
       $this-> name = $data['name'];
       $this-> description = $data['description'];
    }
    public function getName() {
       return $this-> name;
    }
    public function getDescription() {
       return $this-> description;
    }
    public function setName($name) {
       $this-> name = $name;
       $this-> needsUpdating = true;
    }
    public function setDescription($description) {
       $this-> description = $description;
       $this-> needsUpdating = true;
    }
    public function _destruct() {
       if($this-> needsUpdating) {
          $sql = 'UPDATE "widget" SET ';
          $sql .= "\"name\" = '" . pg_escape_string($this-> name) . "', ";
          $sql .= "\"description\" = '" .
```

```
            pg_escape_string($this-> description) . "'";
            $sql .= "WHERE widgetID = " . $this-> id;
            $rs = pg_query($this-> hDB, $sql);
        }
        //We ' re done with the database. Close the connection handle.
        pg_close($this -> hDB);
    }
}
?>
```

سازنده این شی یک اتصال به دیتابیس parts ، از طریق کاربر postgres ایجاد می کند. این اتصال برای استفاده های بعدی در یک متغیر عضو ذخیره شده است. مقدار ID ای که به عنوان پارامتر به سازنده ارسال می شود برای ایجاد یک query به کار می رود که اطلاعات widget مشخص شده با کلید اصلی داده شده را از دیتابیس می آورد. سپس این اطلاعات به متغیر های عضو private نسبت داده می شوند تا بعدا توسط توابع get و set مورد استفاده قرار بگیرند. توجه کنید که اگر هر یک از این کارها اشتباه صورت گیرند، سازنده اعلام خطا می کند.

دو متد دسترسی getName() و getDescription() شما را قادر می سازد تا مقادیر متغیر ها private را بگیرید. به طور مشابه متد های setName() و setDescription() به شما این امکان را می دهند که مقادیر جدیدی را به آنها نسبت دهید. دقت کنید، زمانی که مقدار جدیدی نسبت داده می شود، مقدار needsUpdating به true تغییر می کند. اگر این متغیر تغییر نکند، یعنی هیچ به روز رسانی ای لازم نیست.

برای تست کردن کد بالا، فایلی با عنوان testWidget.php ایجاد کرده و اسکریپت زیر را در آن وارد کنید:

```
<?php
    require_once('class.Widget.php');

    try
    {
        $objWidget = new Widget(1);
        echo 'Widget Name:'.$objWidget->getName().'<br>';

        echo 'Widget Description:'.$objWidget->getDescription().'<br>';
```

```
        $objWidget->setName('Bar');
        $objWidget->setDescription('This is a Bartacular widget!');
    }
    catch (Exception $e)
    {
        die ("There was a problem".$e->getMessage());
    }
?>
```

با اجرای این فایل در مرورگرتان، برای بار اول خروجی مشابه زیر خواهد بود:

Widget Name: Foo

Widget Description: This is a footacular widget!

برای تمامی دفعات بعدی خروجی به صورت زیر خواهد بود:

Widget Name: Bar

Widget Description: This is a bartacular widget!

دیدید که این کد چقدر تکنیک قدرتمندی است. شما شی ای را از دیتابیس می گیرید، PRP آن را تغییر می دهید و به صورت اتوماتیک در دیتابیس ذخیره می شود. تمامی این کار ها تنها با چند خط کد ساده در testWidget.php اتفاق می افتند. اگر تغییراتی صورت نگیرد، شما نیازی به مراجعه به دیتابیس نخواهید داشت و اطلاعات در سرور دیتابیس ذخیره می شوند و این کار عملکرد برنامه ی شما را در مراحل بعد افزایش می بخشد.

کاربران این شی نیازی به دانستن نحوه ی کار کردن آن ندارند. اگر یکی از اعضای با تجربه تیم توسعه کلاس widget را بنویسد، می تواند آن را به دیگر اعضای تازه وارد تیم بدهد و آنها حتی بدون دانستن SQL یا دیگر موارد به کار رفته در کلاس widget می توانند از آن استفاده کنند.

وراثت

اگر شما بخواهید برنامه ای برای دفتر دارایی یک فروشنده ی ماشین بنویسید، احتمالا به کلاس هایی چون Sedan, PickupTruck و MiniVan نیاز خواهید داشت که به ماشین هایی با همان عنوان در ماشین فروشی اشاره کنند. برنامه ی شما نه تنها باید تعداد باقی مانده ی این ماشین ها را نگه دارد، بلکه باید اطلاعات هر کدام از آنها را نیز داشته باشد تا فروشندگان بتوانند این اطلاعات را به خریداران بدهند.

Sedan یک ماشین 4 دره است و شما باید فضای صندلی پشتی و همچنین صندوق عقب را ثبت کنید. PickupTruck صندوق عقب ندارد اما کاروانی با حجم مشخص به آن وصل است. برای یک MiniVan هم شما احتمالا باید لیستی از تعداد درهای کشویی و تعداد صندلی ها را ثبت کنید.

به هر حال هر کدام از این ماشین ها یک نوع اتومبیل بوده و هر کدام تعدادی مشخصه همچون رنگ، سازنده، مدل، سال تولید، شماره ماشین و ... را در برنامه شما خواهند داشت. برای این که مطمئن شوید تمامی کلاس های شما این اطلاعات را دارد شما می توانید از آنها کپی گرفته و در کلاس های خود قرار دهید. همان طور که در ابتدای مقاله گفتیم یکی از مزیت های OOP امکان استفاده ی دوباره از کد های نوشته شده است. بنابرین نه تنها شما نیازی به کپی کردن کدها ندارید، بلکه می توانید از PRP ها و متد های این کلاس ها در یکدیگر با استفاده از مفهوم وراثت استفاده کنید. وراثت توانایی یک کلاس برای استفاده از PRP ها و متد های کلاس والد می باشد.

وراثت شما را قادر می سازد تا یک کلاس پایه با عنوان Automobile ایجاد کنید. سپس شما می توانید بگویید که تمامی کلاس های دیگر همگی نوعی Automobile هستند و تمامی مشخصات و PRP هایی را که یک اتومبیل دارد را دارند. چون Sedan یک اتومبیل است بنابرین به طور اتوماتیک تمامی آنچه در کلاس Automobile تعریف شده است را به ارث می برد، بدون اینکه لازم باشد شما یک خط کد کپی کنید. با این روش شما می توانید تمامی مشخصاتی که

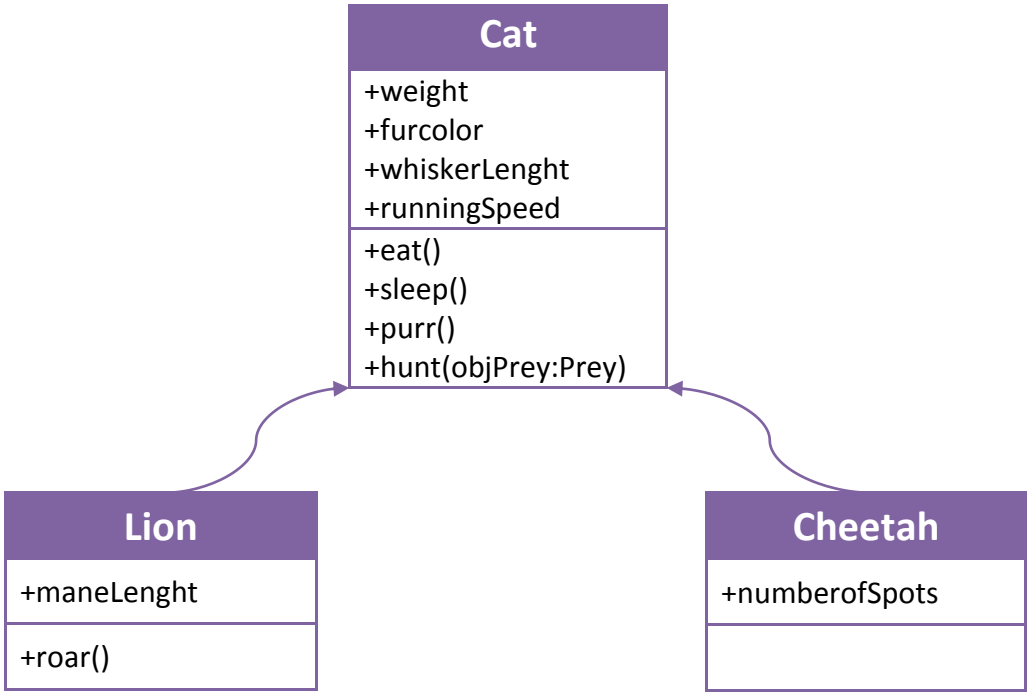
Sedan دارد و در کلاس Automobile نیستند را به این کلاس بیفزایید. به عبارت دیگر، تمام کاری که شما باید انجام دهید تعریف کردن تفاوت ها می باشد و تمامی تشابهات از کلاس اتومبیل به ارث برده شده اند.

توانایی استفاده دوباره از کد ها یکی از مزیت های وراثت می باشد، اما وراثت یک مزیت بزرگ دیگر نیز دارد. فرض کنید کلاسی با عنوان Customer دارید که شامل متد buyAutomobile می باشد. این متد یکی از اشیای کلاس اتومبیل را به عنوان پارامتر دریافت کرده و توابع درون آن تمامی اطلاعات مربوط به آن ماشین را چاپ می کنند. از آنجایی که تمامی Sedan ها، PickupTruck ها و MiniVan ها اتومبیل هستند، شما می توانید اشیای این کلاس ها را به تابعی که منتظر دریافت یک اتومبیل به عنوان پارامتر است، پاس دهید. چون هر سه کلاس گفته شده از یک کلاس کلی تر ارث می برند، شما می دانید که تمامی آنها مجموعه ای از مشخصات و متد های پایه را خواهند داشت. تا زمانی که شما تنها به مشخصات عمومی ماشین نیاز داشته باشید، می توانید کلاس هایی را که از کلاس اتومبیل ارث می برند را به این تابع پاس دهید.

مثال گربه ها را در نظر بگیرید. تمامی آنها خصوصیات مشترک با یکدیگر دارند، مثل خوردن، خوابیدن، خرخر کردن و شکار. علاوه بر این مشخصات مشترک هر کدام از آنها قد، رنگ، طول سبیل و سرعت خاص خود را دارند. اگرچه شیرهای نر یالی با طول مشخص در اطراف گردنشان دارند، نعره می زنند و چیتاها پوستی خال خالی دارند، گربه های معمولی هیچ کدام از این مشخصات را ندارند ولی هر سه این حیوانات از خانواده ی گربه ها هستند.

در php شما می توانید با استفاده از کلید واژه ی extends به php بگویید که این کلاس زیر مجموعه ای از کلاس دیگری است و باید تمامی متد ها و PRP های آن کلاس را به ارث ببرد، و اینکه اگر شما خواستید می توانید توابعی را به آن بیفزایید و یا متغیر های جدیدی برایش تعریف کنید.

اگر بخواهید برنامه ای برای کار با حیوانات باغ وحش بنویسید، احتمالا به کلاس های Cat، Lion و Cheetah نیاز پیدا خواهید کرد. قبل از نوشتن کدها، نمودار کلاس خود را در نظر بگیرید. نمودار کلاس شما باید یک کلاس والد با عنوان Cat داشته باشد که زیرکلاس های Lion و Cheetah از آن ارث می برند. شکل زیر نمودار شما را نشان می دهد:



نمودار کلاس گربه سانان

هر دو کلاس Lion و Cheetah از کلاس Cat ارث می برند، اما کلاس Lion علاوه بر آن PRP ای با عنوان maneLength و متدی به نام roar() را نیز پیاده سازی می کند، در حالی که کلاس Cheetah فقط PRP ای به نام numberofSpots را اضافه کرده است.

کلاس Cat که در فایل class.Cat.php قرار دارد باید به صورت زیر باشد:

```php
<?php
    class Cat {
       public $weight;
       public $furColor;
       public $whiskerLength;
       public $maxSpeed;

       public function eat() {
          //Code for eating ...
```

```
        }

        public function sleep() {
            //Code for sleeping ...
        }

        public function hunt(Prey $objPrey) {
            //Code for hunting objects of type Prey which we'll not define ...
        }

        public function purr() {
            echo 'Purrrrrrrrrr...'.'<br>';
        }
    }
?>
```

این کلاس ساده همه ی PRP ها و متد های معمول گربه سانان را پیاده سازی کرده است. شما می توانید برای ایجاد کلاس های Lion و Cheetah تمام کد بالا را کپی کرده و در آنها قرار دهید و تغییرات لازم را انجام دهید. اما این کار دو مشکل ایجاد می کند.

نخست اینکه اگر شما مشکلی در کلاس Cat پیدا کردید، باید آن را در کلاس های Lion و Cheetah نیز تصحیح کنید. و این باعث ایجاد کار اضافه برای شما می شود.

ثانیا، فرض کنید متدی از یک کلاس دیگر (برای مثال CatLover) دارید که به صورت زیر است:

```
public function petTheKitty(Cat $objCat) {
    $objCat -> purr();
}
```

اگرچه نگه داری شیر یا چیتا به عنوان یک حیوان خانگی اصلا ایمن نیست، اما اگر بتوانید به اندازه ی کافی به آنها نزدیک شوید، خواهید دید که آنها نیز خر خر می کنند. بنابرین شما باید بتوانید شی ای از کلاس Lion یا Cheetah را به تابع petTheKitty() ارسال کنید.

بنابرین شما باید از روش دیگری برای تعریف کلاس های Lion و Cheetah استفاده کنید که همان به کار بردن وراثت می باشد. با استفاده از کلمه ی extends و بعد از آن آوردن اسم

کلاسی که ارث برده است، شما به راحتی می توانید دو کلاس جدید که تمامی ویژگی های کلاس Cat را دارند ایجاد کنید و ویژگی های دیگری را که می خواهید به آنها اضافه کنید.

در این مثال کلاس Lion شما به صورت زیر خواهد بود:

```
<?php
    require_once('class.Cat.php');
    class Lion extends Cat {

       public $maneLength;

       public function roar() {
          echo 'Roarrrrrrrrrr...'.'<br>';
       }
    }
?>
```

با کلاس بالا شما می توانید کارهایی مثل کار زیر را انجام دهید:

```
<?php
    include('class.Lion.php');

    $objLion = new Lion();
    $objLion -> weight = 200; //200 kg
    $objLion -> furColor = 'brown';
    $objLion -> maneLength = 36; //36 cm

    $objLion -> eat();
    $objLion -> roar();
    $objLion -> sleep();

?>
```

همانطور که در این مثال دیدید، شما می توانید PRPها و متد های کلاس والد Cat را بدون نوشتن دوباره ی همه ی آن کدها فراخوانی کنید. به خاطر داشته باشید که واژه ی extends به php می گوید که باید به طور اتوماتیک تمامی PRP ها و متدهای کلاس Cat به همراه تمامی مشخصات کلاس Lion را در برنامه include کند. این واژه همچنین به php می گوید که یک شی از کلاس Lion شی ای از کلاس Cat محسوب می شود و شما اکنون می توانید تابع petTheKitty() را برای آن فراخوانی کنید:

```php
<?php
    include('class.Lion.php');

    $objLion = new Lion();
    $objPetter = new CatLover();
    $objPetter -> petTheKitty($objLion);
?>
```

با این روش هر تغییری که شما در کلاس Cat ایجاد کنید به طور اتوماتیک توسط کلاس Lion به ارث برده می شود. باگ های رفع شده، توابع، متدها و PRP های اضافه شده همگی به تمام زیرکلاس ها ارث داده می شوند.

در مثال بعدی شما خواهید دید که چگونه می توان با استفاده از یک سازنده، یک زیر کلاس را ایجاد کرد. یک فایل جدید با عنوان class.Cheetah.php ایجاد کرده و کد زیر را در آن قرار دهید:

```php
<?php
    require_once('class.Cat.php');

    class Cheetah extends Cat {
       public $numberofSpots;

       public function _construct() {
            $this -> maxSpeed = 100;
       }
    }
?>
```

کد زیر را در فایل testcats.php وارد کنید:

```php
<?php
    require_once('class.Cheetah.php');

    function petTheKitty(Cat $objCat) {
       if($objCat -> maxSpeed < 5) {
            $objCat -> purr();
       }
       else {
            echo 'Can not pet the kitty - it is moving at '.
            $objCat -> maxSpeed.' kilometers per hour!';
       }
    }

    $objCheetah = new Cheetah();
    petTheKitty($objCheetah);

    $objCat = new Cat();
```

```
    petTheKitty($objCat);
?>
```

کلاس Cheetah یک متغیر public جدید با عنوان numberofSpots و یک سازنده دارد که در کلاس Cat نبودند. بعد از این هر وقت شما یک Cheetah جدید بسازید، مقدار پروپرتی maxSpeed که آن را از کلاس Cat به ارث برده است برابر با 100 خواهد بود. دقت کنید، چون در کلاس Cat برای maxSpeed مقدار اولیه ای تعریف نشده است، مقدار آن در تابع petTheKitty() برابر با 0 در نظر گرفته می شود. کسانی که در خانه گربه داشته باشند میدانند که زمان خواب سرعت آنها به 0 میل میکند.

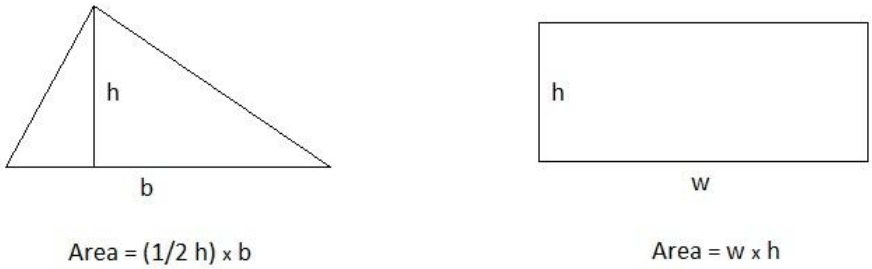
با افزودن توابع، PRPها و یا حتی سازنده ها و مخرب های جدید، زیرکلاس ها می توانند به راحتی با مقدار اندکی کد خود را گسترش دهند و ویژگی های جدیدی را به برنامه شما اضافه کنند. هر زمان که بتوانید کلاسی را یک نوع خاص از کلاسی دیگر تعریف کنید، در این مواقع از وراثت استفاده کنید تا برنامه ی شما حداکثر پتانسیل قدرت برنامه نویسی را داشته باشید.

بازنویسی متدها

چون یک کلاس از کلاسی دیگر ارث برده است، لزومی ندارد که حتما از پیاده سازی والد خود برای یک تابع خاص استفاده کند. برای مثال اگر شما در حال نوشتن برنامه ای برای محاسبه ی مساحت اشکال مختلف هندسی باشید، احتمالا کلاس هایی با عنوان Rectangle و Triangle خواهید داشت. هر دو این شکل ها نوعی چندضلعی هستند و از یک کلاس والد با عنوان Polygon ارث می برند.

کلاس Polygon یک PRP به نام numberOfSides و یک متد به نام getArea خواهد داشت. تمامی چند ضلعی ها مساحتی برای محاسبه دارند، هر چند که روش محاسبه مساحت برای انواع مختلف آنها فرق می کند. فرمول محاسبه مساحت مستطیل برابر است با

w * h که w همان عرض و h همان طول مستطیل می باشند. برای محاسبه مساحت مثلث از فرمول 0.5 * h * b استفاده میشود که در آن h ، ارتفاع مثلث و b همان قاعده ی مثلث می باشد.



برای هر زیرکلاسی که از Polygon می سازید، باید متد معادله پیش فرض محاسبه ی مساحت را با متد معادله ی مربوط به آن شکل عوض کنید. با دوباره تعریف کردن متد محاسبه مساحت شما می توانید پیاده سازی مورد نظر خود را داشته باشید.

در مورد مستطیل، شما باید دو PRP جدید با نام های height و width را اضافه کرده و متد ()getArea را که در کلاس Polygon پیاده سازی شده، بازنویسی کنید. برای کلاس مثلث نیز شما باید PRPهایی را برای ذخیره اطلاعاتی در مورد سه زاویه، ارتفاع و قاعده، تعریف کرده و مند ()getArea را برای آن بازنویسی کنید. با استفاده از وراثت و بازنویسی متدها، شما می توانید به زیر کلاس ها اجازه دهید متد های کلاس والد را آن طور که می خواهند از نو پیاده سازی کنند.

تابعی که یک چندضلعی را به عنوان پارامتر گرفته و مساحت آن را چاپ می کند، به طور اتوماتیک متد ()getArea زیر کلاسی را که به آن پاس داده شده است، فراخوانی می کند. این توانایی زبان های OOP که به صورت اتوماتیک در زمان اجرا تشخیص می دهد که باید کدام متد ()getArea را فراخوانی کند، به چند ریختی یا Polymorphism معروف است. چندریختی، به توانایی برنامه برای انجام کارهای مختلف بر اساس شی ای که بر رویش عمل می کند، می گویند. در این مثال یعنی فراخوانی متدهای ()getArea مختلف بر اساس نوع چندضلعی که به تابع داده می شود.

بعضی مواقع شما می خواهید که پیاده سازی صورت گرفته توسط کلاس والد را حفظ کنید و در عین حال چندین فعالیت دیگر به متد زیرکلاس اضافه کنید. برای مثال اگر شما برنامه ای برای مدیریت یک شرکت غیر انتفاعی داشته باشید، احتمالا کلاسی به نام Volunteer (داوطلب) خواهید داشت که متدی به نام signup() دارد. این متد به داوطلبان اجازه می دهد تا در یک پروژه شرکت کنند و آن کاربر را به لیست داوطلبان آن کار می افزاید.

ممکن است شما، کاربرانی با برخی محدودیت ها همچون زمینه ی جنایی یا ... داشته باشید که باید آنها را از شرکت در برخی پروژه ها منع کنید. در این مورد، چند ریختی به شما این امکان را می دهد تا کلاسی با عنوان RestrictedUser ایجاد کرده و در آن از متد بازنویسی شده ی signup() استفاده کنید، که این متد ابتدا محدودیت های کاربران را بررسی می کند، سپس به آنها اجازه ی شرکت در پروژه را می دهد و یا آنها را منع می کند. اگر محدودیتی نداشته باشند، شما می توانید توابع کلاس والد را برای تکمیل ثبت نام آنها در پروژه، فراخوانی کنید.

زمانی که شما کلاسی را بازنویسی می کنید، نیاز ندارید تمام آن را دوباره بنویسید. شما می توانید از همان پیاده سازی کلاس والد استفاده کنید و فقط مشخصات مورد نظر خود را به آن بیفزایید.

در مثال زیر شما دو کلاس Rectangle و Square را ایجاد خواهید کرد. یک مربع خود نوعی مستطیل به شمار می آید. هر کاری که شما با یک مستطیل انجام می دهید را می توانید با یک مربع نیز انجام دهید؛ اما چون مستطیل دو ضلع متفاوت دارد ولی مربع تنها یک ضلع دارد، باید کارهای متفاوتی را با آنها انجام دهید.

فایل جدیدی با عنوان class.Rectangle.php ایجاد کرده و کد زیر را در آن قرار دهید:

```php
<?php
    class Rectangle {
       public $height;
       public $width;

       public function _construct($width,$height) {
            $this -> width = $width;
            $this -> height = $height;
```

```
        }

        public function getArea() {
             return $this->width * $this->height;
        }
    }
?>
```

این یک روش خوب پیاده سازی، برای مدل سازی مستطیل است. سازنده ی کلاس طول و عرض را به عنوان پارامتر قبول کرده و متد getArea() مساحت آن را محاسبه می کند.

اکنون نگاهی به کلاس class.Square.php بیاندازید:

```
<?php
    require_once('class.Rectangle.php');

    class Square extends Rectangle {

        public function _construct($size) {
             $this -> width = $size;
             $this -> height = $size;
        }

        public function getArea() {
             return pow($this->height , 2);
        }
    }
?>
```

در این کد هم سازنده و هم متد getArea() بازنویسی شده اند. برای اینکه مستطیلی، مربع باشد، باید هر چهار ضلعش به یک اندازه باشند. در نتیجه سازنده ی کلاس تنها به یک پارامتر نیاز خواهد داشت و اگر بیش از یک پارامتر به آن ارسال شود، مابقی را نادیده می گیرد.

اگر تعداد پارامتر های ارسال شده به یک تابع تعریف شده توسط کاربر بیشتر از تعداد پارامتر های خود تابع باشد، php هیچ خطایی را صادر نمی کند. در برخی موارد این یک رفتار مطلوب می باشد. برای اطلاعات بیشتر می توانید به مستندات تابع func_get_args() مراجعه کنید.

اگرچه متد پیش فرض موجود در کلاس Rectangle می توانست به طور صحیح مساحت مربع را نیز محاسبه کند، اما این متد برای افزایش عملکرد برنامه بازنویسی شد، چرا که گرفتن یک PRP و محاسبه ی مجذور آن، سریعتر از گرفتن دو PRP و ضرب آنها در یکدیگر صورت می گیرد.

حفظ عاملیت والد(Preserving the Parnet's Functionality)

گاهی مواقع شما می خواهید تا از توابعی که در کلاس والد پیاده سازی شده اند، با تغییرات اندکی استفاده کنید. در این صورت مجبور نیستید تمام آن را بازنویسی کنید. تنها می توانید تغییرات مورد نظرتان را به آن اضافه کنید.

برای فراخوانی توابع والد از parent::[function name] استفاده کنید. زمانی که شما می خواهید تنها چند تغییر در یکی از متد های والد ایجاد کنید، بعد از فراخوانی آن تابع با parent::[function name] می توانید تغییرات مورد نظر خود را اعمال کنید.

در مثال زیر دو کلاس با نام های Customer و SweepstakesCustomer داریم. یک سوپرمارکت برنامه ای دارد که گاهی مواقع به خاطر برخی مناسبت ها برنامه ی صندوق پول فروشگاه را تغییر میدهد. هر خریداری که وارد فروشگاه می شود برای خودش یک ID دارد که از دیتابیس می آید، و یک شماره مشتری دارد که به تعداد مشتریان قبل اشاره دارد. در شرط بندی های این فروشگاه، خریدار یک میلیونم برنده جایزه خواهد شد.

کلاسی با عنوان class.Customer.php ایجاد کرده و کد زیر را در آن وارد کنید:

```
<?php
    class Customer {
       public $id;
       public $customerNumber;
       public $name;
```

```
        public function _construct($customerID) {
            //Fetch customer information from database
            $data = array();
            $data['customerNumber'] = 1000000;
            $data['name'] = 'Jane Johnson';

            //Assign the values from the database to this object
            $this -> id = $customerID;
            $this -> name = $data['name'];
            $this -> customerNumber = $data['customerNumber'];
        }
    }
?>
```

اکنون فایلی به نام class.SweepstakesCustomer.php ایجاد کرده و کد زیر را در آن قرار دهید:

```
<?php
    require_once('class.Customer.php');

    class SweetstakesCustomer extends Customer {

        public function _construct($customerID) {
            parent::_construct($customerID);

            if ($this->customerID == 1000000)
            {
                echo 'Congratulations, You win a year supply of stakes!';
            }
        }
    }
?>
```

وراثت چگونه کار می کند؟

کلاس customer مقادیر اولیه را با استفاده از customer ID از دیتابیس می گیرد. شما باید مقدار customer ID را از کارت هایی که در اکثر فروشگاه های زنجیره ای بزرگ به مشتریان داده می شود، بگیرید. با استفاده از customer ID شما می توانید اطلاعات شخصی

مشتریان را به همراه عددی که تعداد مشتریان قبل از او را نشان می دهد از دیتابیس بگیرید. تمامی این اطلاعات را در متغیر های عمومی (public) ذخیره کنید.

کلاس SweepstakesCustomer چند کار دیگر را به سازنده می افزاید. شما ابتدا باید با استفاده از parent::_construct ، سازنده ی والد را فراخوانی کرده و پارامتر هایی را که می خواهد به آن بدهید. سپس به مشخصه ی customerNumber نگاه می کنید و اگر این مشتری، شماره یک میلیونوم باشد به او می گویید که برنده شده است.

برای این که ببینید این کلاس چگونه کار می کند، فایلی با نام testCustomer.php ایجاد کرده و کد زیر را در آن قرار دهید:

```
<?php
    require_once('class.SweepstakesCustomer.php');
    //since thid file already includes class.Customer.php, there's
    //no need to pull that file in, as well.

    function greetCustomer(Customer $objCust) {
       echo "Welcome back to the store $objCust->name!";
    }

    //change this value to change the class used to create this customer
    //object
    $promotionCurrentlyRunning = true;
    if($promotionCurrentlyRunning) {
       $objCust = new SweepstakesCustomer(1000000);
    }
    else {
       $objCust = new Customer(1000000);
    }
    greetCustomer($objCust);
?>
```

در ابتدا مقدار $promotionCurrentlyRunning را برابر با false قرار دهید و فایل را در مرورگرتان باز کنید. زمانی که مقدار $promotionCurrentlyRunning را true کنید، پیغامی مبنی بر برنده بودن آن شخص چاپ می شود.

واسط ها

در برخی از مواقع، شما کلاس هایی را دارید که رابطه ی بینشان وراثت نیست. ممکن است کلاس های کاملا متفاوتی داشته باشید که فقط بعضی از رفتار هایشان مشترک است. برای مثال یک شیشه ی مربا و یک در را در نظر بگیرید. هر دوی این اشیا می توانند باز و بسته شوند، اما هیچ ربطی به هم ندارند. فرقی نمی کند که چه نوع در یا شیشه ای باشد، آنها این صفات مشترک را با هم دارند، اما به جز باز و بسته شدن هیچ اشتراک دیگری با هم ندارند.

واسط ها چه کار می کنند

شما این جنبه را در OOP نیز دیده اید. یک واسط (Interface) شما را قادر می سازد تا مشخص کنید که فلان شی می تواند فلان کار را انجام دهد، اما نحوه ی انجام آن را به شما نشان نمی دهد. یک واسط، قراردادی بین اشیای بی ربط برای انجام کار های معمول می باشد. شی ای که این واسط را پیاده سازی می کند به تمامی اعضای خود این ضمانت را میدهد که می تواند تمامی کارهایی که در واسط تعریف شده اند را انجام دهد. دوچرخه ها و لوازم فوتبال کاملا اشیای متفاوتی هستند، اما اشیایی که نمایانگر آنها را در یک مغازه ی لوازم ورزشی هستند، باید بتوانند با هر دوی آنها کار کنند.

با تعریف یک واسط و پیاده سازی آن در اشیایتان می توانید کلاس های کاملا متفاوتی را به توابع معمول ارسال کنید. مثال زیر همان مساله ی شیشه ی مربا و در را نشان می دهد. فایلی با عنوان interface.Openable.php ایجاد کنید:

```php
<?php
    interface Openable {
       public function open();
       public function close();
    }
?>
```

همانطور که برای نام گذاری کلاس هایتان به صورت class.[className].php عمل می کردید، برای واسط ها نیز بهتر است از همان روش استفاده کنید. یعنی به صورت زیر آنها را نام گذاری کنید: interface.[interface Name].php

شما واسط Openable را با استفاده از همان روشی که کلاس ها را تعریف می کنید، تعریف کردید. به جز اینکه به جای class از interface استفاده کرده اید. یک واسط هیچ متغیر عضوی ندارد و هیچ کدام از توابع عضو خود را پیاده سازی نمی کند. چون هیچ پیاده سازی صورت نگرفته، شما باید این توابع را به صورت انتزاعی یا Abstract تعریف کنید. این کار به php می گوید که هر کلاسی این واسط را پیاده سازی کند، خودش مسئول پیاده سازی توابع می باشد. اگر نتوانید تمامی توابع واسط را در آن کلاس پیاده سازی کنید، php یک خطای زمان اجرا خواهد داد. شما نمی توانید توابعی را که می خواهید پیاده کنید، بلکه باید تمامی آنها را پیاده سازی کنید.

واسط ها چگونه کار می کنند؟

واسط Openable قراردادی بین اعضای یک برنامه است که می گوید هر کلاسی که می خواهد از این واسط استفاده کند، باید دو تابع open() و close() را در خودش پیاده سازی کند. با این قرارداد شما می توانید از توابع نام برده برای اشیای مختلفی استفاده کنید، بدون اینکه هیچ ارتباط وراثتی بین آنها وجود داشته باشد.

فایل زیر را ایجاد کرده و نام آنرا class.Door.php بگزارید:

```
<?php
    require_once('interface.Openable.php');

    class Door implements Openable {
       private $_locked = false;

       public function open() {
            if($this->_locked) {
                echo 'Can not open the door. It is locked.';
```

```
            }
            else {
                echo 'creak ...<br>';
            }
        }
        public function close() {
            echo 'Slam!<br>';
        }
        public function lockDoor() {
            $this->_locked = true;
        }
        public function unlockDoor() {
            $this->_locked = false;
        }
    }
?>
```

اکنون فایلی با عنوان class.Jar.php ایجاد کنید:

```
<?php
    require_once('interface.Openable.php');

    class Jar implements Openable {
        private $contents;

        public function _construct($contents) {
            $this->contents = $contents;
        }
        public function open() {
            echo 'The jar is now open! <br>';
        }
        public function close() {
            echo 'The jar is now closeed! <br>';
        }
    }
?>
```

برای استفاده از این فایل ها، فایل دیگری با عنوان testOpenable.php در همان پوشه ایجاد کنید:

```
<?php
    require_once('class.Door.php');
    require_once('class.Jar.php');
```

```php
    function openSomething(Openable $obj) {
       $obj->open();
    }

    $objDoor = new Door();
    $objJar = new Jar("Jam");

    openSomething($objDoor);
    openSomething($objJar);
?>
```

چون هر دو کلاس Door و Jar واسط Openable را پیاده سازی کرده اند، شما می توانید هر دوی آنها را به تابع openSomething() ارسال کنید. چون این تابع فقط اشیایی را که واسط Openable را پیاده سازی کرده اند می پذیرد، در نتیجه شما می توانید توابع open() و close() را در آن فراخوانی کنید. اما شما نمی توانید در داخل کلاس openSomething() به PRP کلاس Jar که همان contents می باشد و یا به متد های کلاس Door که lock() و unlock() می باشند، دسترسی داشته باشید؛ چراکه این PRPها و متد ها جزئی از واسط نیستند.

کپسول سازی

همان طور که قبلا گفته شد، اشیا شما را قادر می سازند تا جزئیات پیاده سازیشان را از دید کاربران مخفی نگه دارید. شما برای فراخوانی متد signup() از کلاس Volunteer که قبلا گفته شده نیازی ندارید بدانید که این کلاس داده هایش را در دیتابیس ذخیره می کند یا در یک فایل text معمولی یا در فایل XML یا هر چیز دیگری. به طور مشابه لازم نیست بدانید که آیا اطلاعات داوطلبان که به عنوان اشیایی از کلاس volunteer هستند به صورت متغیر های جداگانه تعریف شده اند، یا به صورت آرایه و یا حتی به صورت اشیای دیگر.

این توانایی مخفی کردن جزئیات پیاده سازی به encapsulation یا کپسول سازی معروف است. به عبارت دیگر کپسول سازی شامل دو مفهوم زیر می باشد:

- محافظت داده های داخلی کلاس از کد های خارجی
- مخفی کردن جزئیات پیاده سازی

معنای لغوی encapsulation قرار دادن در کپسول یا یک پوشش خارجی می باشد. یک کلاس خوب پوسته ی خارجی کاملی را به دور محتویات داخلیش می کشد و واسطی را تعریف می کند تا با استفاده از آن کد های خارج کلاس به صورت کاملا جدا از محتویات خود کلاس، نوشته شوند. با این کار شما دو مزیت زیر را به دست می آورید:

- شما می توانید جزئیات پیاده سازی کلاستان را، هر زمان که خواستید بدون تغییر دادن کدهایی که از کلاستان استفاده می کنند تغییر دهید.
- چون شما می دانید هیچ چیزی از خارج کلاس نمی تواند مقادیر PRPها و state ها را تغییر دهد، شما می توانید به آن مقادیر همیشه اعتماد کنید.

متغیر های عضو و متدهای هر کلاس یک امکان دید یا visibility دارند. visibility به آنچه که می تواند از بیرون کلاس دیده شود، اشاره می کند. همانطور که قبلا گفته شد، اعضا و متد های private تنها برای خود کلاس قابل استفاده هستند. اعضا و متدهای protected برای خود کلاس و زیر کلاس های آن قابل دسترسی هستند. و در نهایت از اعضا و متدهای public در همه جا می توان استفاده کرد.

به طور کلی بهتر است تمامی متدها و اعضای کلاس را private کنید و هر گونه دسترسی به این اعضا و متدها باید از طریق متدهای دسترسی یا (accessor methods) صورت گیرد. شما به کسی اجازه نمی دهید ، زمانی که چشمانتان بسته است به زور غذایی را به شما بخوراند، بلکه شما می خواهید تا آن را بررسی کرده و سپس تصمیم بگیرید که آیا می خواهید آن را بخورید یا نه؟ به طور مشابه زمانی که از کپسول سازی استفاده می شود، شما می توانید تغییراتی

را که توسط یک تابع public بر روی شی شما صورت می گیرد را بررسی کرده و آنها را پذیرفته یا رد کنید.

برای مثال اگر شما در حال نوشتن برنامه ای برای یک بانک باشید که با جزئیات حساب کاربران کار میکند باشید، احتمالا شی ای به نام Account با PRP ای به نام totalBalance و متدهای ()makeDeposit و ()makeWithdrawal خواهید داشت. مشخصه ی Balance باید به صورت read-only یا فقط خواندنی باشد. تنها راهی که می توان موجودی (balance) را تغییر داد از طریق برداشت (withdrawal) یا واریز (deposit) می باشد. اگر totalBalance به صورت public باشد، شما می توانید کدی بنویسید که بدون انجام تراکنش واریز موجودی حساب افزایش یابد، که این اصلا برای یک بانک خوب نیست.

در عوض شما باید این PRP را به صورت private تعریف کرده و متدی با عنوان ()getTotalBalance را ایجاد کنید که مقدار این عضو private را برگرداند. چون متغیری که مقدار totalBalance را نگه می دارد به صورت private است، شما نمی توانید آنرا به طور مستقیم تغییر دهید و از آنجایی که تنها دو متد ()makeWithdrawal و ()makeDeposit می توانند بر موجودی تاثیر بگذارند، شما مجبورید برای افزایش موجودی حسابتان اقدام به واریز پول کنید.

با توانایی مخفی کردن جزئیات پیاده سازی و حفاظت از متغیر های عضو، یک توسعه نرم افزاری شی گرا به شما امکان می دهد تا برنامه ای منعطف و پایدار (stable) بنویسید.

تغییرات شی گرایی در PHP

از php 3 به بعد امکان پشتیبانی از اشیا در php قرار داده شد. در ابتدا هدف اصلا پشتیبانی از مفهوم امروزی کلاس ها و اشیا نبود، بلکه به صورت بسیار محدود پشتیبانی می شدند.در ابتدا هدف پشتیبانی از اشیا به خاطر یک روش مناسب و ساده برای گروه بندی داده ها و

توابع بودن بود. با همه گیر شدن php مفهوم شی گرایی نیز در آن توسعه یافت و استفاده از آن در برنامه های بزرگ بسیار معمول شد.

در واقع هیچ پشتیبانی از کپسول سازی وجود نداشت. شما نمی توانستید متغیرها یا متدهایتان را به صورت private یا protected تعریف کنید. همه چیز به صورت public بود. همانطورکه دیدید این امر باعث ایجاد مشکلاتی می شود.

علاوه بر آن هیچ پشتیبانی از متدها و واسط های انتزاعی وجود نداشت. متدها و متغیرها نمی توانستند به صورت static تعریف شوند. هیچ مخربی تعریف نشده بود. عدم وجود این موارد باعث می شد تا ترجمه برنامه ها از زبان های دیگر مثل C++ یا Java به php مشکل شود. جدول زیر برخی از ویژگی های جدید php 5 و php 6 را نشان می دهد:

ویژگی جدید	مزیت
متغیرها و متدهای private و protected	کپسول سازی و حفاظت از داده ها اکنون در php امکان پذیر است.
پشتیبانی از ارجاع های چند مرحله ای	استفاده از جملاتی مثل $obj->getSth()->doSth() هم اکنون در php امکان پذیر است.
متغیرها و متدهای static	تعریف متدهایی که می توانند به صورت استاتیک فراخوانی شوند، در php اضافه شده است.
سازنده های یکپارچه	هم اکنون تمامی سازنده ها به صورت __construct() تعریف می شوند.
پشتیبانی از مخرب ها	با استفاده از متد __destruct() ، کلاسهای php از امکان تخریب برخوردار شده اند. این ویژگی این امکان را می دهد تا به هنگام تخریب یک شی کارهایی را که می خواهید انجام دهید.
پشتیبانی از کلاس ها و واسط های انتزاعی	شما می توانید متد های را در کلاس والد تعریف کرده و آنها را در زیرکلاس ها پیاده سازی کنید.
نوع پارامترها	شما می توانید یک کلاس را به عنوان پارامتر برای توابعی که شی می گیرند ، پاس دهید. مثل function foo (Bar $objBar) {...} که به شما این اطمینان را می دهد که پارامتر از همان نوعی است که شما می خواستید.

خلاصه

در این مقاله شما با مفهوم برنامه نویسی شی گرا (Object Oriented Programming) آشنا شدید. ما کلاس را به صورت یک نقشه برای ایجاد اشیا نشان دادیم. اشیا مجموعه ای از داده ها و توابعی هستند که بر اساس تعریف کلاس ساخته می شوند. هر کدام از اشیا مشخصاتی دارند که PRP یا property نامیده می شود. همچنین آنها رفتارهایی نیز دارند که متد یا method نامیده می شوند. می توانید PRPها را متغیر و متدها را تابع فرض کنید.

برخی از کلاس ها از یک کلاس والد سهم می برند. برای مثال مربع ها (فرزند) خود نوعی مستطیل (والد) هستند. زمانی که شما کلاسی را به عنوان یک زیر کلاس از کلاسی دیگر تعریف می کنید، تمامی متدها و PRP های آن را به ارث می برد. شما می توانید متدهای کلاس والد را بازنویسی کنید و یا از همان پیاده سازی والد استفاده کنید و فقط چند ویژگی دیگر به آن اضافه کنید. و یا اینکه اصلا متد را بازنویسی نکنید و دقیقا از متد والد استفاده کنید.

کپسول سازی مفهوم مهمی در OOP می باشد. کپسول سازی به توانایی یک کلاس برای حفاظت از محتویاتش در مقابل دیگران گفته می شود. متدها و PRP های هر کلاس سه سطح visibility دارند:

- public
- private
- protected

اعضای private تنها برای خود کلاس قابل دسترسی هستند. اعضای protected برای خود کلاس و زیرکلاس های آن قابل دسترسی هستند و اعضای public برای همه در دسترس می باشند.

با معرفی php 5 و Zend Engine 2 ، پشتیبانی از شی گرایی در php تغییرات زیادی پیدا کرد. ویژگی های جدید و بهبود های قابل توجهی که در php 5 صورت گرفته و بعد از آن در php 6 توسعه یافته است، php را به یک زبان شی گرای کامل تبدیل کرده است.

About the Authors

Ed Lecky-Thompson was a founding partner of Brandspace before starting his own interactive agency in 2003. Currently, he heads up U.K. digital specialists Galileo (www.galileodm.com), where he runs thriving digital relationship marketing and online public relations accounts for top blue chips (including Microsoft and a major U.S. financial services group). He has written several books on PHP over the past five years, as well as writing for php|architect magazine. Ed was nominated for a New Media Age Effectiveness Award in 2004 for his work with First Leisure Corporation as head of new media.

Steven D. Nowicki is a senior software developer at AdKnowledge, Inc., and has more than 13 years of experience with software development and technology management in New York, London, and Los Angeles. He has been the lead software architect for several multimillion-dollar Web applications and dozens of large-scale, mission-critical PHP implementations, including enterprise resource planning, CRM systems, and high-volume analytics. This is his third book on PHP.

Thomas Myer is a technical book author, consultant, and Web developer. In 2001, he founded Triple Dog Dare Media in Austin, Texas. Triple Dog Dare Media focuses its attention on helping companies create CodeIgniter-based applications such as content management, portals, and eCommerce systems. He is the author of No Nonsense XML Web Development with PHP (Sitepoint, 2004) and Lead Generation on the Web (O'Reilly, 2007). His newest book is Professional CodeIgniter (Wrox, 2008). He has also authored dozens of technical and business articles for IBM DeveloperWorks, Amazon Web Services, AOL, Darwin Magazine, and others.

About the interpreter

Since 2009 I'm studing Information Technology in Institute for Advanced Studies in Basic Sciences (IASBS).

E-mail : masoudmanson@gmail.com
Cellular phone : 09148401824
Blog : iasbs-it.mihanblog.com

Masoud Amjadi