**Expanding on OOP**

The simple OOP example we created isn't necessarily the best way to create a class and use it. In that example, you created all of the properties using the public classification. This means that any application that uses the Product class can directly access the properties and modify them with whatever values it wants. That could be dangerous, and it's somewhat frowned upon in OOP circles. What if a wayward application set the price property of a product to a negative value? OOP allows you to be able to code your classes to help prevent accidents like that from happening.

The preferred way to handle properties in a class is to make them private so any external code can't change them. It then creates public methods that allow programs to set and get the property values. You can control exactly what happens to the properties in those public methods. Let's create another example that demonstrates this technique:

1. Create a file called *example2.php* in the oop folder.
2. Open the file in a text editor, and add the following code:

<?php

class Product {

private $description;

private $price;

private $quantity;

public function setDescription($value) {

$this->description = $value;

}

public function getDescription() {

return $this->description;

}

public function setPrice($value) {

if ($value > 0)

$this->price = $value;

else

$this->price = 0;

}

public function getPrice() {

return $this->price;

}

public function setQuantity($amount) {

$this->quantity = $amount;

}

public function getQuantity() {

return $this->quantity;

}

public function printProduct() {

echo "Product: $this->description<br>\n";

printf("Price: $%.2f<br>\n", $this->price);

echo "Quantity: $this->quantity<br>\n";

}

public function buyProduct($val) {

$this->quantity -= $val;

}

}

$prod1 = new Product();

$prod1->setDescription("Carrot");

$prod1->setPrice(1.50);

$prod1->setQuantity(10);

echo "Just added product:<br>\n";

$prod1->printProduct();

echo "<br>Buying 4 carrots.\n";

$prod1->buyProduct(4);

echo "Quantity is now: " . $prod1->getQuantity() . "<br>\n";

?>

1. Save the file, and exit the text editor.
2. Open a browser and go to the URL: *http://localhost/oop/example2.php*. You should see the following output in your browser window:

Results of the example2.php code

Notice that the new methods set the property values inside the class. Only they're allowed to access the properties directly. If you try using the code:

$prod1->quantity = 10;

in your application, you'll get an error message, since the properties are now private. This provides a basic level of protection for your data. You can put any type of checks in the methods that set property values. The setPrice() method demonstrates this feature by checking the value that the program code assigns to the price property.

There's one downside to protecting your class properties, though. Now it's somewhat of a pain to create a new class object, because you have to use the individual set methods to define each property value. Fortunately, PHP provides an easy way to solve this problem.

**Class Constructors**

Most OOP languages provide a special method called a *constructor*. The constructor allows you to automatically define property values when you instantiate an object. PHP uses the special method \_\_construct() to define the constructor:

public function \_\_construct($name, $value, $amount) {  
$this->description = $name;  
if ($value > 0)  
$this->price = $value;  
else  
$this->price = 0;  
$this->quantity = $amount;  
}

The constructor for our Product class uses three parameters to assign values to the three class properties when you instantiate an object. Now, to create a new Product object, you must use this format:

$prod1 = new Product("Carrot", 1.50, 10);

You can now use a constructor to force the program code to set the class properties when it instantiates the object. If you choose, you can also block any properties from changing by not providing methods that change the values. That's a lot of control over your code!

Let's play with this concept a little. Follow these steps to build both a class file and an application file:

1. Create the file *Product.inc.php* in the oop folder.
2. Open the file in a text editor, and add the following code:

<?php

class Product {

private $description;

private $price;

private $quantity;

public function \_\_construct($name, $value, $amount) {

$this->description = $name;

if ($value > 0)

$this->price = $value;

else

$this->price = 0;

$this->quantity = $amount;

}

public function setDescription($value) {

$this->description = $value;

}

public function getDescription() {

return $this->description;

}

public function setPrice($value) {

if ($value > 0)

$this->price = $value;

else

$this->price = 0;

}

public function getPrice() {

return $this->price;

}

public function setQuantity($amount) {

$this->quantity = $amount;

}

public function getQuantity() {

return $this->quantity;

}

public function printProduct() {

echo "Product: $this->description<br>\n";

printf("Price: $%.2f<br>\n", $this->price);

echo "Quantity: $this->quantity<br>\n";

}

public function buyProduct($amount) {

$this->quantity -= $amount;

}

public function addProduct($amount) {

$this->quantity += $amount;

}

}

?>

1. Save that file, and exit the text editor.
2. Now create another file called *example3.php* in the oop folder.
3. Open that file in a text editor, and enter the following code:

<?php

include("Product.inc.php");

$prod1 = new Product("Carrots", 1.50, 10);

echo "Just added product:<br>\n";

$prod1->printProduct();

$prod2 = new Product("Onions", 2.00, 15);

echo "<br>Just added product:<br>\n";

$prod2->printProduct();

echo "<br>Buying 4 carrots.\n";

$prod1->buyProduct(4);

$quant = $prod1->getQuantity();

echo "Quantity is now: $quant<br>\n";

echo "Buying 3 onions.\n";

$prod2->buyProduct(3);

$quant = $prod2->getQuantity();

echo "Quantity is now: $quant<br>\n";

echo "Adding 10 more carrots.\n";

$prod1->addProduct(10);

$quant = $prod1->getQuantity();

echo "Quantity is now: $quant<br>\n";

?>

1. Save the file, and exit the text editor.
2. Open a browser and go to the URL: *http://localhost/oop/example3.php*. Your output should look similar to this:

Output of the example3.php code

The Product.inc.php file contains the class definition code for the Product class. Putting the class definition code in a separate file is a common practice that allows you to use the class in any PHP program you want. Naming the file the same as the class name isn't necessary, but it sure makes life easier! All you have to do is use the include() function to bring the class definition code into your application code. Once you include the class definition, you can use it to create as many class objects as you need to handle your data.

In the next chapter, you'll see how to extend OOP classes and use that technique to get fancy with accessing the MySQL server.