**Extending Classes**

No, I'm not talking about trying to make your course longer! OOP provides a way to extend an existing class by adding additional functionality. That's the whole beauty of OOP. You can take classes and use them as-is, or you can modify just the pieces you need to fit a situation.

Defining a new class that's an extension of another class is called *inheritance*. The new class (called the *child*) inherits all of the members of the original class (called the *parent*). You can then add new members to the child class and even overwrite members of the parent class. If you use the overwritten members, the child members take precedence over the parent members.

To create a class extension, you use the class definition format, along with the *extends*keyword and the name of the class you're extending:

class Soda extends Product {  
.  
}

The new class, Soda, automatically contains all of the class members of the Product class. Thus, you can assign values to the $quantity property using the setQuantity() method in Soda, just as you did with the Product class. You can also create new class members to add to the extended class in the class definition. Let's create another example and try this:

1. Create a file called *example4.php* in the oop folder.
2. Open the file in a text editor, and add the following code:

<?php

include("Product.inc.php");

class Soda extends Product {

private $ounces;

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

parent::setDescription($name);

parent::setPrice($value);

parent::setQuantity($amount);

$this->ounces = $size;

}

public function printProduct() {

parent::printProduct();

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

}

}

$prod1 = new Soda("Root Beer", 1.25, 10, 18);

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

$prod1->printProduct();

echo "<br>Buying 5 bottles.\n";

$prod1->buyProduct(5);

echo "now there's " . $prod1->getQuantity() . " left<br>\n";

$prod1->printProduct();

?>

1. Save the file, and exit the text editor.
2. Open a browser and go to the URL: *http://localhost/oop/example4.php*.

When you view the Web page in your browser, it should look like this:

The output of the example4.php file

Because the Soda class extends the Product class, you must use the include() function to include the Product class definition code before you can write the class extension code. The class Soda extends the Product class by adding a new property ($ounces). Also, it overwrites two existing method members of the Product class: the constructor and the printProduct() method.

Both the constructor and printProduct() methods use another odd feature of OOP. The double colon symbol (its official name is the *scope resolution operator*) allows access to the parent properties and methods from the child class. In this example, I use the *get* series of methods from the Product class to assign values to the properties contained in the parent. To reference the properties in the parent, you must specifically point to the parent methods using the scope resolution operator:

parent::setDescription($name);

You use the same technique to print the parent properties using the printProduct() method. The only thing the child printProduct() method must add is print the $ounces property.

**Using OOP With Your Database**

As you've seen throughout this course, the PHP MySQL extension provides lots of functions for interacting with a MySQL server. However, all of these functions use the procedural style of programming. If you're using OOP to create your application, you won't want to mix OOP code with procedural code just to connect with your database.

To resolve this problem, PHP includes another extension package, the MySQL Improved extension (called php\_mysqli). The php\_mysqli extension provides both OOP classes and procedural functions for connecting to the database, submitting queries, and retrieving result sets. The great thing is that you can use the php\_mysqli extension as a parent class and create your own customized methods for submitting queries and retrieving your data. This is where the real power of OOP comes into play!

|  |
| --- |
| **Note:** To use the php\_mysqli extension, you must have it installed and activated on your AMP server. The WampServer already has this extension installed and activated. If you're following along on another server, follow the procedure for activating an extension on your server. |

The MySQL Improved extension uses the PHP class called *mysqli*. The basic format for connecting to a MySQL database, submitting a query, and retrieving the result set using the mysqli class is:

$con = new mysqli("localhost", "test", "test", "store");

$query = "SELECT description, price, quantity FROM products";

$result = $con->query($query);

while( $row = $result->fetch\_assoc())

{

$description = $row['description'];

$price = $row['price'];

$quantity = $row['quantity'];

}

The process is similar to the mysql\_ functions we've used in this course but with an OOP twist.

**Combining It All Together**

Now let's really take a dive into the OOP world. For the last example, you'll extend the mysqli class to create your own database methods that are specialized to your application environment. First, you need to create your new class, called ProductDatabase:

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

<?php

include("Product.inc.php");

class ProductDatabase extends mysqli {

public function clean\_and\_query($query) {

if (get\_magic\_quotes\_gpc())

$query = stripslashes($query);

$query = $this->real\_escape\_string($query);

return $this->query($query);

}

public function getProduct($result) {

if ($row = $result->fetch\_assoc())

{

$prod = new Product($row['description'], $row['price'], $row['quantity']);

return $prod;

} else

{

return FALSE;

}

}

}

?>

1. Save the file, and exit the text editor.

By default, your ProductDatabase class inherits all of the standard methods and properties of the mysqli class, so we'll be able to use the query() and fetch\_assoc() methods as normal. This class creates a new method, called *clean\_and\_query()*. The purpose of this specialized class is to remove any added slashes by the magic\_quotes\_gpc feature, and then it uses the mysqli version of the mysql\_real\_escape\_string() function to add any slashes to trouble characters.

The getProducts() method is even trickier. It receives the result of a mysqli query, then processes it in a mysqli fetch\_assoc() method. The trick is it puts the returned result in a Product class object for you. This lets you use the methods you created in the Product class to process your product data. Awesome!

Now you can create an application that uses your new class. Follow these steps:

1. Create a file called *example5.php* in the oop folder.
2. Open the file in a text editor, and add the following code:

<?php

include("ProductDatabase.inc.php");

$con = new ProductDatabase("localhost", "test", "test", "store");

$query = "SELECT description, price, quantity FROM products";

$result = $con->clean\_and\_query($query);

while ( $product = $con->getProduct($result))

{

$product->printProduct();

echo "----------------------<br>\n";

}

?>

1. Save the file, and exit the text editor.
2. Open a browser and go to the URL: *http://localhost/oop/example5.php*

You should see the output of all the records in your products table.

The output of the example5.php file

To use your new class, you must instantiate a new object using the ProductDatabase() constructor. Since it inherits the mysqli class, you can use the same constructor format. Then, use your new clean\_and\_query() function to submit your database query, and send the results to your new getProducts() method. Then the code uses the printProduct() method to easily display the product information.

It's a great way to build your code library with classes that you can use over and over again. Now, let's go on to Chapter 5 and wrap up this lesson.