A Simple PHP Shopping Cart

by Joey Mingrone


Comment on this article

As you may have guessed from the title, in this article we will discuss a method for creating a very simple shopping cart using PHP. The only assumption made is that you have at least a basic understanding of PHP programming. We will first look at object-oriented PHP, so if you already feel comfortable with this topic, then you may skip on to the section titled "A Shopping Cart".

Object-oriented PHP

Whether or not you've programmed in an object-oriented programming language before, object-oriented PHP is simple enough that you can learn to use objects in your scripts in a matter of minutes.

Object-oriented PHP, like any object-oriented language, involves using classes to define objects, and then using object instances to complete a task. The main purpose of a class is to provide instructions that define the functionality of an object. So, the class Apple would provide instructions on the capabilities of apples. Let's look at the definition of a class called Vehicle.

```
class Vehicle {

  var $license;  // these are the instance variables
  var $color;

  // this is the constructor
  function Vehicle($license=NULL, $color=NULL) {
     $this->color = $color;
  }

  // the remaining functions are member functions
  // to set or retrieve the state of the object

  function getColor() {
     return $this->color;
  }

  function getLicense() {
     return $this->license;
  }

  function setColor($color) {
     $this->color = $color;
  }

  function setLicense($license) {
     $this->color = $license;
  }

}
```

The class Vehicle has two instance variables ($license and $color), a constructor (function Vehicle()), and four member functions (all the other

functions). The instance variables describe properties of a vehicle object. The constructor is used to create new vehicle objects. Constructors of a class always have the same name as the class. Here in the vehicle class, the constructor can take two optional parameters to create a new vehicle. The parameters are optional because they are each given a default value (null in our case) in the definition of the constructor. The four member functions either get or set the values of the instance variables. Here is an example of how to create and use vehicle objects:

```
// create a new vehicle.. new calls the constructor
$myVehicle1 = new Vehicle();
$myVehicle2 = new Vehicle(null, "green");
// use the member function setColor()
$myVehicle1->setColor("black");
// show the state of the vehicle objects
echo("myVehicle1 color is " . $myVehicle1->getColor()
     . "<br>myVehicle2 color is "
     . $myVehicle2->getColor() . "<br>");
```

If we wanted to extend the class Vehicle by including more specific information about the vehicle, we could **extend** the class.

```
class Car extends Vehicle {

  var $autoTrans;               // new instance variables
  var $solarPowered;

  function Car($license=null, $color=null,
              $autoTrans=false, $solarPowered=true) {
    $this->license = $license;
    $this->color = $color;
    $this->autoTrans=$autoTrans;
    $this->solarPowered=$solarPowered;
  }

  // new member functions

  function getAutoTrans() {
     return $this->autoTrans;
  }

  function getSolarPowered() {
     return $this->solarPowered;
  }

  function setAutoTrans($autoTrans) {
     $this->autoTrans = $autoTrans;
  }

  function setSolarPowered($solarPowered) {
     $this->solarPowered = $solarPowered;
  }
}



class Truck extends Vehicle {

  var $extendedCab;        // new instance variables
  var $diesel;

  function Truck($license=null, $color=null,
              $extendedCab=false, $diesel=false) {
     $this->license = $license;
     $this->color = $color;
     $this->extendedCab = $extendedCab;
     $this->diesel=$diesel;
  }

   // new member functions

  function getExtendedCab() {
     return $this->extendedCab;
  }

  function getDeisel() {
     return $this->diesel;
  }
```

```
  function setExtendedCab($extendedCab) {
     $this->extendedCab = $extendedCab;
  }

  function setDeisel($diesel) {
     $this->diesel = $diesel;
  }
}
```

Here, we have created two subclasses of Vehicle, Car and Truck. Both Car and Truck **inherit** the instance variables and member functions of Vehicle, and they add their own. Car has added the instance variables $autoTrans and $solarPowered, while Truck has added $extendedCab and $diesel. The new member functions get and set the values of the new instance variables. Here's how we could use these two new classes:

```
// use Car's constructor to create a new car
$myCar = new Car("abc-123", "blue", false, true);
echo("myCar liscense is: " . $myCar->getLicense() . "<br>myCar color is: "
     . $myCar->getColor() . "<br>");
if( $myCar->getAutoTrans() )  // does the car have automatic transmission?
   echo("myCar is an automatic transmission.<br>");
else
   echo("myCar is a standard transmission.<br>");
if( $myCar->getSolarPowered() )  // is the car solar powered?
   echo("myCar is solar powered.<br>");
else
   echo("myCar is not solar powered.<br>");

echo("<br>");

// use Truck's constructor to create a new truck
$myTruck = new Truck("def-345", "red", true, true);
echo("myTruck license is: " . $myTruck->getLicense()
   . "<br>myTruck color is: "
   . $myTruck->getColor() . "<br>");
if( $myTruck->getExtendedCab() ) // does the truck have an extended cab?
   echo("myTruck has an extended cab.<br>");
else
   echo("myTruck does not have an extended cab.<br>");
if( $myTruck->getDiesel() ) // is the truck a diesel?
   echo("myTruck is a diesel.<br>");
else
   echo("myTruck is not a diesel.<br>");
```

A Shopping Cart

Ok, now that you understand the basics of object-oriented PHP, on to the shopping cart. Let's start right from the beginning. First, what is a shopping cart? Well, if you ever bought anything on-line, the steps you took to make your purchase probably went something like this:

- you browsed or searched through some sort of product catalogue
- you put the products you wanted to buy in your **cart**
- before you made any purchases you reviewed the products in your **cart** and made any necessary changes
- finally, you entered your personal information and purchased the items in your **cart**

So, like the thing you push around in the grocery store, an **online shopping cart** simply stores products you would like to purchase.

First, let's think about what should be stored in a shopping cart. Well, the only essential information is an entry for each product the user has put in her/his cart (in the form of a unique id) along with the quantity. So, a simple cart might look something like this:

ProductID	Quantity
19034	11
453	3

811	1

What are the basic operations that must be performed on the cart? Well, we definitely have to be able to add and remove products. It would also be nice to be able to display the contents. Now that you understand how to use PHP objects let's create a new cart class.

```
class Cart {

  // holds products with quantities
  var $addedProducts;

  // add $qty product(s) to the cart
  function add($id, $qty) {
     if($qty > 0)
        $this->addedProducts[$id] += $qty;
  }

  // remove $qty product(s) from the cart
  function remove($id, $qty) {
     if($qty > 0)
        if($this->addedProducts[$id] >= $qty)
           $this->addedProducts[$id] -= $qty;
  }
}
```

This class contains one instance variable - $addedProducts. This variable is an associative array where the key is the product id and the value is the quantity. So, with our cart above $addedProducts[453] would be 3. The two member functions allow us to add/remove 'x' number of products to/from the cart.

Displaying the contents of the cart requires retrieving more information about the products from the product catalogue database. Since this varies from database to database, you could **extend** the basic cart class to provide the display method. Here's an example. (I've used mysql functions to access the database. If you use a different database supported by PHP, just substitutes its function calls for the mysql calls.)

```
class MyCart extends Cart {

  function display() {
     $ids = "";  // holds all the product ids in the cart
     foreach($this->addedProducts as $id => $qty)
        $ids .= ",$id";
     $ids = substr($ids, 1); // remove the starting ,
     // connect to the database
     if( ($dbLink = dbConnect("host", "user", "password", "database")) < 0 )
        return -1;
     // query the database
     $Query = "SELECT * FROM Products WHERE ProductID IN (" . $ids . ")"
     if( !($productsQID = mysql_query($Query)) )
        return -2;
     // if there were no products found
     if( !(mysql_num_rows($productsQID) > 0) )
        return -3;
     while( $productRow = mysql_fetch_array($productQID) )
        echo($productRow["Description"]  ..etc);
  }
}
```

In the display method, we are putting every Product ID from the Cart into a string and passing that string on to a database query. The query retrieves more information about each Product whose ID is in the string parameter. The while loop is simply going through each Row returned from the query and displaying the product information.

So we know how to create a cart. Now, there's only one small challenge. How do we associate the cart with a user? HTTP (the protocol the web uses) cannot help us because it's stateless. This means HTTP does not provide a means to remember data between various connections. For example, if you go to www.iticentral.com, the web server there processes your request by sending you a web page. After it has finished with your request, it forgets any information it knew about you. This is where PHP's session handling can help. With PHP's session handling, you can preserve data between visits to various web pages. Session handling works by distinguishing a visitor's session with a unique id (a session id). This unique id is stored on the client's system through a cookie or it is propagated in the URL for future requests. Using the session id, variables can be stored on the server and associated with the user's session. This means it is possible to maintain a cart during a visit to a web site.

So, in a nutshell, to create a simple shopping cart using PHP, create a cart object and register it as a session variable.

```
$cart = new MyCart();
session_register("cart");
```

That's it. You have yourself a shopping cart. Now users navigating your site have an easy way to store the products they may wish to purchase. The cart will be accessible in session scripts through $HTTP_SESSION_VARS["cart"] (or simply $cart if you have the register_globals option turned on in the php.ini file). When they are ready to make their purchases, you can gather their personal and financial information and use it with their cart to complete the transaction.

Developed Under:

Apache 1.3.12
PHP Version 4.0.1
MySQL Version 9.38 Distribution 3.22.32 for FreeBSD 4.0