# PHP2

## Class 2 – Week 2 Inheritance, Visibility and Static Functions

**Inheritance**

Remember that classes are ways of grouping similar portions of code together. This means that classes contain code highly related to a task you want to accomplish. Inheritance is the ability for a class to “extend” the functionality of another class, or inherit functionality from another parent class.

In many programming languages a class can inherit or extend from many other classes however, in PHP we can only inherit from a single class (note that the parent class can inherit from another class an so on).

Consider the following illustrations:

1. Actors and Set décor on a stage. What attributes are shared/common amongst these objects? What elements are different/unique. (actor, set object, light, explosive effect, trap-door)
2. What about game design? What are common attributes amongst all enemies? What elements are different?
3. What about a form and form elements on a page? Form element, Date Input
4. What about considering an automobile: Car? Tesla? Truck?

A common example for understanding inheritance is:

MALE & FEMALE = People

This example illustrates that People can be broken down into sub classes each with their own properties.

Consider the following example:

CAR & TRUCK = Vehicles

Notice that the vehicles can inherit similar properties and function from a common class or object description called Vehicles. Vehicles may have the following properties and functions:

Vehicles:

* NumberDoors
* NumberOfSeats
* Accessories
* TopSpeed
* FuelSource
* LicensePlateNumber
* Function StartVehicle()
* Function StopVehicle()
* Function Accelerate()
* Function Decelerate()
* Etc

Now, every type of vehicle that extends from the vehicles object, can inherit and use/call these properties and methods of the vehicle class. Why would we do this? Well, perhaps our vehicles need custom or specific properties and methods that are not shared with a common class. Consider our Truck properties that are unique:

Truck:

* GateHeight
* BedLength
* BedWidth
* CabType
* TowingCapacity
* Etc

It would violate the rules of DRY if we were to place all our methods from the vehicle class into the truck class (and certainly this would be even more problematic if the vehicle class were to be shared with many many other classes!). Thus, we inherit the functionality from Vehicles into Truck.

Inheritance in PHP uses the keyword *extends*.

Class Vehicle

{

var numberDoors;

function startVehicle()

{

echo “started”;

}

}

Class Truck extends Vehicle

{

var towingCapacity;

}

// instantiate an instance of the vehicle class

$myTruck = new Truck();

In the above example you’ll see that we have extended functionality from the parent Vehicle into the Truck class. Now, in the next example we can call the startVehicle() method from the Truck class. Because this method is inherited, we can call I as though it is a built in method of the Truck class.

Class Vehicle

{

var numberDoors;

function startVehicle()

{

echo “started”;

}

}

Class Truck extends Vehicle

{

var towingCapacity;

}

// instantiate an instance of the vehicle class

$myTruck = new Truck();

$myTruck->start();

Now we can interact with the myTruck object, setting and getting and calling methods that exist in either the Truck class or the Vehicle class.

class Cart

{

function sayHello(){

echo “hello world”;

}

function start(){

$this->sayHello();

}

}

// instantiate an instance of the class

$cart = new Cart();

$cart->start();

Do you see how $cart->start(); calls the start method, and that method then calls the sayHello method which is contained within this same class? That will be an important process to understand as it is very common for classes to use many methods inside of itself to accomplish tasks. This lends to the *do no repeat your self* coding philosophy where reusable chunks of code are made into methods.

**Visibility of Methods and Variables**

Visibility. The visibility of a property, a method or (as of PHP 7.1.0) a constant can be defined by prefixing the declaration with the keywords public, protected or private. ... Members declared protected can be accessed only within the class itself and by inheriting and parent classes.

The reason for visibility can be to control or validate a variable before setting and using, showing the intent of your code, and in some ways forces classes to be designed with great intent and forthought to where and how the variables will be used.

There are three levels of visibility:

* public (by default)
* private
* protected

Read this article: <https://torquemag.io/2016/05/understanding-concept-visibility-object-oriented-php/>

**Accessing a Class Without Instantiation**

In PHP we can call methods within a class without instantiating an object for that class. This may be because we have created a class that contains utility functions that we want to call without having to load an object with all the other methods.

Calling a static class also allows us to quickly access snippets of code inside a class.

To call a static function we use the classname::methodName() pattern. We also need to make sure we add the keyword *static* to the class method

class DB

{

public static Connect()

{

…..

}

}

// instantiate an instance of the class

DB::Connect();

Notice now that the Connect() method will be called but no object is every instantiated.

If we want to call a static method from within a class, we can use the *self*  keyword.

class DB

{

public static Connect()

{

self::SelectDB();

…..

}

public static SelectDB()

{

….

}

}

// instantiate an instance of the class

DB::Connect();

As we have not talked about variable visibility, this variable is accessible outside the class, from the instantiated object as well. Thus we can set the variable manually by saying $cart->myName = “Bob”;

Read: <http://www.zentut.com/php-tutorial/php-static/>

**Extending Classes – Overriding Methods**

There may be certain situations when you want to override a parent class (a class you’ve inherited from) function.

class person

{

protected function set\_name($new\_name) {

if ($new\_name != "Jimmy Two Guns") {

$this->name = strtoupper($new\_name);

}

}

}

class employee extends person

{

protected function set\_name($new\_name) {

if ($new\_name == "Stefan Sucks") {

$this->name = $new\_name;

}

}

}

What about if you want to call the inherited method again? Use the keyword

Parent::methodName

OR

Person::methodName;

**Final Methods and Classes**

You can stop entire classes or methods from being overridden via “final class classname{ }”

final class Member {

  // This class can't be extended at all

}

**Referencing Objects**

Referencing objects allows you to pass an object into a class for use. When the class is destroyed the referenced object remains. This allows a referenced object to be used in multiple classes.

Think About:

1. Database connection needed throughout and app
2. User account in an app
3. Think MarioKart

$array = array(1,2,3,4);

foreach ($array as &$value){

$value = $value + 10;

}

unset ($value); // Must be included, $value remains after foreach loop

print\_r($array);

///// COMPARE AGAINST NO REFERENCE////

$array = array(1,2,3,4);

foreach ($array as $value){

$value = $value + 10;

}

unset ($value); // Must be included, $value remains after foreach loop

print\_r($array);

DB Connection Method: NOTE: This is deprecated example but illustrated the point.

class dbConnection

{

var $strUseDB = "CarPartsCompany\_DB";

Function getDataFromDB($strMyQuery)

{

// some db query here…

echo "got data from: ".$this->strUseDB."<br>";

}

Function writeDataToDB($strMyQuery)

{

// some db query here…

echo "wrote some data to: ".$this->strUseDB."<br>";

}

}

Class inventoryList

{

var $oDB;

Function inventoryList(&$oDB)

{

$this->oDB = $oDB;

$arrData = $this->oDB->getDataFromDB("get some data");

}

function showWhichDBUsing()

{

echo "using db: ".$this->oDB->strUseDB."<br>";

}

}

Class receiveOrder

{

var $oDB;

Function receiveOrder(&$oDB)

{

$this->oDB = $oDB;

$this->oDB->writeDataToDB("save some data");

}

function showWhichDBUsing()

{

echo "using db: ".$this->oDB->strUseDB."<br>";

}

}

$oDB = new dbConnection();

$oInventoryList = new inventoryList($oDB);

$oReceiveOrder = new receiveOrder($oDB);

$oDB->strUseDB = "AirPlanePartsCompany\_DB";

$oInventoryList->showWhichDBUsing();

$oReceiveOrder->showWhichDBUsing();

NOTE: Variables are now natively passed by reference into a class. This is important, as modifying the variable within a class will modify it elsewhere.