Introduction to Object-Oriented Programming

Object-Oriented Programming

In this section we'll cover:

- Classes vs. objects – Definitions vs. instances
- Properties – Variables within an object
- Methods – Functions within an object
- Visibility and Scope – An object's right to privacy
- Inheritance – It's classes all the way down!
- Static Members – Using a class without an object
- Abstraction – Defining requirements of a class
- Traits – Reusable code

Classes vs. Objects

- A **class** is a definition of an object. It defines what an object is, what its members are and how it behaves during its lifetime
  - Member: an individual property, method, etc. inside the class
- An **object** is an instance of a class. It's a variable that contains a copy of the class, which can then be changed

Classes vs. Objects (Cont.)

- A Ball class defines the properties of a ball: size, color, diameter, what happens when you kick it, etc.
- A ball object is an instance of the Ball class. The Ball class can be used to “create” a baseball, basketball, kickball, etc by changing the class' properties
- Do you have the balls to build your own class?

Creating A Class

```
// Our class has one property, which stores the ball's color as plain text

class Ball {

    public $color = “red”;

}
```

Creating A Class (Cont.)

- **Members** are the components of a class. When a class is instantiated, you can access the members using [ObjectName]**->**[MemberName].
  - For example, *$basketball->color = "Orange";* sets the color of the basketball object to orange
- **$this** is a special keyword that lets an object refer to itself
  - Remember not to use the object name when inside the class! Would you say "I'm wearing a blue shirt" or "the instance of me is wearing a blue shirt?"

Creating An Object

- **new** is a special keyword that creates an instance of an object. You can pass values (called parameters) to a new object and access them in the object's *constructor*
  - The constructor is a special function that runs when an object is created. PHP has several special functions

Creating An Object (Cont.)

```
// Include the PHP file that defines the class
include 'Ball.php';


// Create an instance of the Ball class and store it as an object in the myBall variable
$myBall = new Ball();


// Print the color of the ball (“red”)
print $myBall->color;
```

Properties

- A **property** is a variable defined in a class. Properties can be created empty or can be *initialized* with a value
- Properties can be accessed from inside and outside the class but are typically accessed using *getters* and *setters*
  - Getters return the value of a property and setters apply a value to a property. This lets you add restrictions, access control, and formatting rules to a property

Properties (Cont.)

```
class Ball {

    // Stores “red” when object is created
    private $color = “red”;

    // Getter for $color. Returns “red”
    public function getColor() {
        return $this->color;
    }
}
```

Methods

- A **method** is a function defined in a class
- Methods can accept multiple values as input, return values as a variable, or modify properties in a class
- Methods can act on the class itself using the **$this** or **self** keywords

Methods (Cont.)

```
class Ball {
    public function getSize($diameter) {
        return “This ball is “ . $diameter . “ inches wide.”;
    }
}

$myBall = new Ball();

// Prints “This ball is 10 inches wide.”
print $myBall->getSize(10);
```

Visibility

- **Visibility** determines which parts of the script can access the class's members
  - *Public*: Up for grabs. Anyone can access
  - *Private*: Can only be accessed from within the same class
  - *Protected*: Can be accessed from within the class and by *parent* or *inherited* classes

Visibility (Cont.)

```
$myBall = new Ball();

// Throws a tantrum. Remember, Ball::color is private!
print $myBall->color;

// ...but getColor is public, so it works
print $myBall->getColor();
```

Scope

- **Scope** is the area in which a single variable can be accessed
  - *Global*: PHP has a single scope for each running script
  - *Local*: Declared within a function and has its own values
  - If a local variable overrides a global variable, the global variable can be accessed using the **global** keyword

Scope (Cont.)

```
$basketText = “Basket”;
$ballText = “ball”;

function printText() {
    global $basketText;
    return $basketText . $ballText;
}

// Only prints “Basket”
print printText();
```

Inheritance

- **Inheritance** lets one class *extend* another
- An extended class is called the *parent*; the extending class is called the *child*
- Children inherit the members of the parent. Children can also access the *protected* and *public* members of their parent class
- Make sure you include the parent class in your script before extending it! Otherwise, PHP will complain about a missing class

Inheritance (Cont.)

```
// Kickball.php
include 'Ball.php';
class Kickball extends Ball {
    private $size = 10;

    public function getSize() {
        return $this->size;
    }
}

$myKickballl = new Kickball();

// prints “10, red”;
print $myKickball->getSize() . “, “ . $myKickball->getColor();
```

Static Members

- Static members are accessed through the class instead of through an object
  - Accessed using [ClassName]**::**[Member]
  - Instead of **this** , the **self** keyword is used to self-reference

Static Members (Cont.)

```
class Volleyball extends Ball {
    private static $manufacturer = “Wilson”;

    public static function getManufacturer() {
        return self::$manufacturer;
    }
}

// Prints “Wilson”
print Volleyball::getManufacturer();
```

Abstraction

- An **abstract** class can't be instantiated
- Abstract classes lay out the requirements of the class, which are then *implemented* in a regular class
- Abstract classes can still provide pre-defined properties and methods
- Abstract classes and members are defined using the *abstract* keyword
- Derived classes *extend* the abstract class

Abstraction (Cont.)

```
abstract class Sphere {
    abstract public function getColor();
    public function getCircumference($radius) {
        return round(2 * 3.14159 * $radius);
    }
}

class Ball extends Sphere {
    private $color = “red”;
    public function getColor() {
        return $this->color;
    }
}

$myBall = new Ball();

// Prints “red, 10”
print $myBall->getColor() . ', ' . $myBall->getCircumference(1.592);
```

Traits

- A **trait** is a block of reusable code used across multiple classes
- Traits define an executable block of code that gets *used* by classes
- The order for precedence for identical traits is class → trait → inherited class

Traits (Cont.)

```
trait BallPhysics {
    public function Bounce() {
        print “Boing!”;
    }
}

class Ball {
    use BallPhysics;
}

$myBall = new Ball();

// Prints “Boing!”
print $myBall->Bounce();
```

Happy Hacking!

More resources:

PHP: Classes and Objects (php.net)

Object Oriented PHP for Beginners (KillerPHP)

PHPaaSA Message Board (Meetup)