ROBOT OOP

Learning the basics of Object Oriented Programming using robots from popular culture
GOOD SOFTWARE

• Highly cohesive
• Loosely coupled
IN THE BEGINNING

• Before OOP there was Procedural
• A procedure is series of steps - like a recipe
• We use functions to organize our code
• functions are used by many languages
WHY OOP

• It’s a major part of modern programming
• Not knowing it will hurt your career
• Every CMS And Framework uses it
• It enables you to write better code
OVERVIEW
The Basics of Objects
WHAT IS AN OBJECT

• an **object** is an instance of a class

• an **instance** is a single occurrence of something
WHAT IS A CLASS

• A class specifies the object’s internal data and representation and defines the operations the object can perform
WTF

While those definitions are technically correct, they’re not very helpful
A Class is a blueprint.
It defines what the object is and what it can do.

An Object is a class made real.
It's a bundle of data and behavior.
INSTANTIATION

• The act of creating an instance of a class

```php
<?php
include "class.robot.php";

$first = new Robot();
$second = new Robot();
```
POPULAR WORDPRESS CLASSES

- WP_Query
- WP_Rewrite
- WP_Error
- WP_Widget
4 PRINCIPLES OF OOP

• Abstraction
• Encapsulation
• Inheritance
• Polymorphism
ABSTRACTION

separation from details
INTERFACES

• Details aren’t important to the User

• Desktop is an interface

• USB is an interface
FUNCTIONS

• name
• parameters
• return values
• provide scope for variables
CLASS

- It’s the basis for OOP in PHP
- Scope for data members and methods
- Abstraction from main program
- Can be used between programs and projects
INSIDE THE CLASS

- member variables (data members)
  - instance variables
  - class (static) variables
- methods
  - instance methods
  - static methods
SNEAK PREVIEW

- Encapsulation
- Inheritance
- Polymorphism
- Composition
- Type Hinting
- Interfaces
ENCAPSULATION

Hiding the details
BASICS

• Sometimes called *Information Hiding*

• Scope

• Visibility in classes
METHODS

• They’re just functions
• clear names
• function scope provides protection
• limited activity
VISIBILITY

• You set visibility to prevent *unauthorized* changes

  • **Public** - everyone can access
  • **Protected** - you and your relatives
  • **Private** - Just for you
SHOW US THE CODE
class Person {
    private $firstname = null;
    
    public function get_name()
    {
        return $this->firstname;
    }
    
    // ... Lots of other stuff would go here
}
INHERITANCE
SIX

Intelligent, Cunning, and Alluring … also a Cylon
WHAT IS A CYLON?

Cybernetic Lifeform Node
class Cylon
{
    public function __construct()
    {
        $this->created = new DateTime();
    }

    public function __destruct()
    {
        error_log( 'Died at ' . new DateTime() );
        $this->download();
    }

    /** @todo Add some cool artificial intelligence */
}
EXTENDING CLASSES

• uses the extends keyword

• gets everything from it’s parent

• then adds it’s own data members and methods
class Six extends Cylon {
    // add methods here
}

WHY INHERITANCE

- It allows you to easily re-use code
- It’s a way to organize related classes
- Write less code
TYPE HINTING

• Used when defining functions or methods
• Specify what class a parameter must be
• public function get_name( Cylon $cylon )
ABSTRACT CLASSES

• Still provides core functionality for child classes

• **Not directly instantiated**

• Allow your descendants chance to change
abstract class Cylon
{
    public function __construct()
    {
        $this->created = new DateTime();
    }

    public function __destruct()
    {
        echo 'Died at ' . new DateTime() . "\n" ;
        $this->download();
    }

    /** @todo Add some cool artificial intelligence */
}
CONTROL IT

• It creates a tight relationship between classes

• Not too deep - limit to 2 levels

• Can make it hard to move class to new project
COMPOSITION

Let’s form Voltron
COMPOSITION

• One object is a part of another
• Uses the public interface
• Preferred over inheritance
• Modular and loosely coupled
class Person {
    private $firstname = null;
    private $dateOfBirth = null;

    public function __construct()
    {
        $this->dateOfBirth = new DateTime();
    }

    public function getName()
    {
        return $this->firstname;
    }

    public function getDateOfBirth()
    {
        return $this->dateOfBirth->format('c');
    }
}
class Lion {
    private $color = "";

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

    public function form() {
        printf("%s Lion!\n", $this->color);
    }
}

class Voltron {

    public function __construct( $black_lion ) {
        $this->head_torso = $black_lion;
    }

    public function form(){
        $this->left_leg->form();
        $this->right_leg->form();
        $this->left_arm->form();
        $this->right_arm->form();
        $this->head_torso->form();
    }

}
POLYMORPHISM
from the Greek, meaning “many forms”
TYPE OF POLYMORPHISM

• Sub-Type polymorphism aka inheritance
• function overriding - redefined by subclass
• function overloading - not supported by PHP
FUNCTION OVERLOADING

- two methods w/same name but different signatures
- Java does this. Not supported by PHP.
class Person{

    public function setName( $first, $last ){
        $this->firstName = $first;
        $this->lastName = $last;
    }

    public function setName( $fullname ){
        list($first, $last) = explode(' ', $fullname);
        $this->firstName = $first;
        $this->lastName = $last;
    }

}
INTERFACES

- use **implements** not **extends**
- Have no functionality - just names and parameters
- Useful across unrelated classes.
interface iOpenClose
{
    public function open();
    public function close();
}

class File implements iOpenClose {
}

class Door implements iOpenClose {
}
interface iDestroyHumanity
{
    public function destroy();
}

class Cybermen implements iDestroyHumanity {
}

class Dalek implements iDestroyHumanity {
WRAPPING UP
4 PRINCIPLES OF OOP

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism
FURTHER STUDY

• Design Patterns
• Principles of S.O.L.I.D.
• Collections
• Iterators
• Exceptions
S.O.L.I.D. PRINCIPLES

• Single Responsibility Principle
• Open Closed Principle
• Liskov Substitution Principle
• Interface Segregation Principle
• Dependency Inversion Principle
HOMEWORK

• Create a WordPress Widget for a sidebar
• Create a class to parse an RSS feed
• Create a WordPress Plugin using OOP
FUN STUFF

• Form Voltron  https://www.youtube.com/watch?v=tZZv5Z2Iz_s

• Battlestar Galactica

• Doctor Who, S01 E06 - “Dalek”

• Doctor Who, S02 E05 - “Rise of the Cybermen”
THANK YOU

• Andrew Woods

• @awoods on Twitter

• http://andrewwoods.net