OO and Ahh!

An Introduction to Object Oriented Programming With PHP

John Valance
division 1 systems
johnv@div1sys.com

About John Valance

- Independent Consultant
  - Founder and CTO of Division 1 Systems (div1sys.com)
  - Specialty is helping IBM shops develop web applications and related skills
  - Training, mentoring, project management, consultation and coding
- 30+ years IBM midrange experience (S/38 thru IBM i)
- 13+ years of web development experience
  - Web scripting language of choice = PHP
- Frequent presenter on web development topics
- Trainer for Zend Technologies
  - Teaches Intro to PHP for RPG programmers
  - Zend Certified Engineer

Goals and Topics of This Presentation

- Goals:
  - Introduce Object Oriented programming concepts and basic OO syntax for PHP
  - Focus on basics – avoid advanced OO concepts
- Topics:
  - Review of PHP functions – concepts and syntax
  - Basic concepts, keywords and syntax
    - Defining classes and instantiating objects
  - Examples
    - Person class
    - HTML form input class

Assumptions

- You:
  - Understand basic PHP syntax
  - Understand PHP functions
  - Understand basics of web programming in PHP
  - Some experience with HTML forms and PHP
  - Interested in Object-Oriented PHP
  - May have no prior experience with OO
    - maybe you've tried it, but got lost or overwhelmed

Review of PHP Functions

- Functions have several important properties that set them apart from RPG subroutines
  - Parameters = input
  - Return value = output
  - Local variables i.e., scope
- Functions are defined with function keyword

```
2  function formatDate( $dateString, $format = 'M d, Y' )
3  {
4      $dateVar = strtotime($dateString);
5      $fmtDate = date($format, $dateVar);
6      return $fmtDate;
7  }
8  // Mainline
9  echo formatDate('2011-09-27'); // "Sep 27, 2011"
10 echo formatDate('2011-09-27', 'm/d/y'); // "09/27/11"
```

Anatomy of `formatDate()` function

- Two parameters, passed by VALUE (i.e. copy)
- $format has default value (optional parameter)
- `$dateString`, `$format`, `$dateVar`, and `$fmtDate` are all LOCAL variables

  (i.e., not accesible outside formatDate function)

```
2  function formatDate( $dateString, $format = 'M d, Y' )
3  {
4      $dateVar = strtotime($dateString);
5      $fmtDate = date($format, $dateVar);
6      return $fmtDate;
7  }
8  // Mainline
9  echo formatDate('2011-09-27'); // "Sep 27, 2011"
10 echo formatDate('2011-09-27', 'm/d/y'); // "09/27/11"
```

Functions are Reusable

- Function = Black-Box
  - Pre-tested component
  - Well-defined interface (API)
  - Can be relied upon as building block

- Can create function libraries and include them in multiple scripts
  - Use require_once('func_lib.php');

Why Objects?

Why not just use functions, and organize them into libraries?

- Objects take the concepts of functions a step further (a big step).
- Objects allow you to organize your functions into groups that share a common set of data elements

Encapsulation:

- Data and Related Functions are tied together

What is a Class?

- A class is a template for creating objects
- Defines:
  - a set of related data elements
  - a set of functions that perform actions on this data
- Data elements are called “Properties”
  - these are PHP variables
- Actions are called “Methods”
  - these are PHP functions

Person class

```php
<?php
class Person {
    // Data Elements (i.e.: properties)
    public $firstName;
    public $lastName;

    // Functions (i.e.: methods)
    public function __construct( $first, $last ) {
        $this->firstName = $first;
        $this->lastName = $last;
    }

    public function getFullName() {
        $fullName = $this->firstName . ' ' . $this->lastName;
        return $fullName;
    }

    public function sayHello() {
        echo "Hello, my name is " . $this->getFullName();
    }
}
```

Defining a Class

- Use class keyword, followed by name of the class
- Body of class:
  - curly braces { } enclose entire class definition
  - Properties (variables)
  - Methods (functions)
- Properties usually coded at top, before methods

```
class ClassName {
      // properties…

      // methods…
}
```

Class Names

- Class name should be a noun
  - represents an object of some sort that we are attempting to model
- Standard is to start with capital letter
  - use mixed case or underscores to separate multiple words in class name.
- Some examples of classes you might create:
  - Customer, Order, Product, HTML_InputForm, DB2_Connection, ErrorLog, etc.

Basic OO Design

- Anything that can be boiled down to
  - a set of properties (variables)
  - and actions (functions) that can be performed on the properties

  can be modeled as an object class.

- Typically code each class in a separate PHP file
  - Use same name as class for file name: Person.php
- Include class def into applications
  - use require_once() function

Using Classes in Applications

CREATING AN OO APPLICATION

Instantiating Objects

- Class definitions are just template
  - by themselves won't accomplish anything or run any code.
- Need to create an object instance to use them
- Use the “new” keyword, followed by Class name
- Assign this to an object variable

```
$bob = new Person;
$tom = new Person;
```

- Each object has its own set of properties

Person class again

```php
<?php
class Person {
    // Data Elements (i.e.: properties)
    public $firstName;
    public $lastName;

    // Functions (i.e.: methods)
    public function __construct( $first, $last ) {
        $this->firstName = $first;
        $this->lastName = $last;
    }

    public function getFullName() {
        $fullName = $this->firstName . ' ' . $this->lastName;
        return $fullName;
    }

    public function sayHello() {
        echo "Hello, my name is " . $this->getFullName();
    }
}
```

Application using Person class

personApp.php

```php
<?php
require_once 'Person.php';

$bob = new Person('Bob', 'Smith');
$tom = new Person('Tom', 'Jones');

$bobsName = $bob->getFullName();
echo "Bob's full name is $bobsName <br />";

$tom->sayHello();
```

If your run this script in a browser, it will produce the following output:

Bob's full name is Bob Smith
Hello, my name is Tom Jones

Class vs. Object

- Similar to the relationship between a File Description and a Record
  - Class is like a DB File Layout
    - Except a Class also defines functions that can act upon the data.
  - Object is like a record in the file
    - A single “instance” of the data
- DB anology ends there
  - no database is involved
  - objects are in memory while script executes
- Each object has its own set of variables
- Each object is an “instance” of the class
  - Instantiation is done with the “new” keyword

Object Member Access: ->

- Properties and Methods are "Members" of the class
- Access to object members done with "->"
  - object member access operator
  - aka, arrow operator

```
echo 'First name: ' . $bob->firstName;
$tom->sayHello();
```

- Note: from outside the class definition, can only access ***public*** members.

Using $this Within a Class

- From within class definition, accessing properties and methods *of the same class* is done using **`$this`**
- **$this** is a special object name
  - **can only be used inside class methods (not from application code)**
  - **refers the current instance of the class, based on context in which method was called**

Example of $this - In Context

- Application context ($tom object):

```
$tomsName = $tom->getFullName();
```

- Class context ($this object):

```
public function getFullName() {
    $fullName = $this->firstName.' '.$this->lastName;
    return $fullName;
}
```

- When `getFullName()` called on $tom object, **$this** refers to $tom's data

Constructors

- When object instantiated, special method called __construct( ) is called automatically
  - 2 underscores before construct in the name
- In OO languages, this is called a *constructor* method.
- Performs object initialization tasks
  - like RPG's *INZSR subroutine
- Optional – don't have to code __construct()

Constructor Example

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

- Constructor can be coded with parameters
  - e.g., to set property values

```
$bob = new Person('Bob', 'Smith');
$tom = new Person('Tom', 'Jones');
```

Member Visibility: public vs. private

- Every class member (property/method) should specify the **"visibility"**
- **public** visibility:
  - member can be directly accessed from any context, inside or outside the class definition
- **private** visibility:
  - member can only be directly accessed from the methods within the same
- Default (implicit) visibility is public
  - but you should explicitly specify visibility for each member

Person class – one more time

```php
<?php
class Person {
    // Data Elements (i.e.: properties)
    public $firstName;
    public $lastName;

    // Functions (i.e.: methods)
    public function __construct( $first, $last ) {
        $this->firstName = $first;
        $this->lastName = $last;
    }

    public function getFullName() {
        $fullName = $this->firstName . ' ' . $this->lastName;
        return $fullName;
    }

    public function sayHello() {
        echo "Hello, my name is " . $this->getFullName();
    }
}
```

Getters and Setters

- Public properties are considered bad form
  - If public, you can retrieve and modify values
  - Public ties you to a specific implementation
- Best practice:
  - make all properties private
  - declare public getter and setter methods to access properties
- Aka – “accessor” methods
  - controls access to object data

Benefits of Accessor Methods

- Setters: Add filtering and error checking on values before allowing data elements to be set.
  - setFirstName() method could check for a maximum length
  - throw an error if the value supplied for firstName will not fit into a database field.
- Getters: Format data value for consumption by a variety of applications before being retrieved.
  - getFirstNameForWeb() could filter value using PHP's htmlentities() function, preventing XSS attacks.
- Can simulate read-only properties
  - Property has public getter, but no public setter
- Abstracts the Interface from Implementation
  - Changes to the implementation do not affect applications

Person Class – Private Properties

```php
class Person {
    private $firstName;
    private $lastName;

    public function __construct($first, $last) {
        $this->setFirstName($first);
        $this->setLastName($last);
    }
    public function setFirstName($first) {
        if ($this->checkFirstName($first)) {
            $this->firstName = $first;
            return true;
        }
    }
    public function getFirstName() {
        return htmlentities($this->firstName);
    }
    private function checkFirstName($first) {
        if (strlen ( $first ) > 40) {
            throw new Exception('firstName exceeds 40 characters' );
        } else {
            return true;
        }
    }
```

Error Handling in OO Code

```
private function checkFirstName($first) {
    if (strlen ( $first ) > 40) {
        throw new Exception('firstName exceeds 40 characters' );
    } else {
        return true;
    }
}
```

- We don't know in which context an object will be used
- In class, if error occurs, throw an Exception
- It will bubble up through call stack until caught
- Allows application code to handle error appropriately
- Uncaught exceptions will cause fatal error, and produce ugly stack trace on web page.

Try / Catch Blocks

```
try {
    // try block: i.e., code accessing
    // objects which may throw exception
} catch (Exception $e) {
  // catch block: i.e., code to execute
  // if exception thrown in try block
}
```

personApp.php, revised:

```
try {
    $tom->setFirstName('Thomas');
} catch (Exception $e) {
    echo "An error occurred as follows: " . $e->getMessage();
    echo "<br>Stack Trace: <br>" . $e->getTraceAsString();
}
```

Exception Class

http://www.php.net/manual/en/class.exception.php

- **Exception** is a PHP built-in class

```
throw new Exception('firstName exceeds 40 characters' );
```

- This instantiates an unreferenced object of type Exception

```
catch (Exception $e)
```

- This receives the thrown Exception object, and assigns it to a variable named $e
- We can now access the public members of the Exception object via the variable $e

```
echo "Error occurred: " . $e->getMessage();
```

Example: `Form_Input` class

- Create a Class to:
  - store the properties of an HTML form input field
  - render the HTML for the <input> tag, in a variety of formats
- Properties:
  - name (attribute of <input> tag)
  - type (attribute of <input> tag)
  - value (attribute of <input> tag)
  - text label (to display next to the input field)
  - output only? (boolean: true = protect input)
- Methods:
  - constructor (parms: name - req'd.; type - optional, default='text')
  - setters/getters for private properties
  - render (returns HTML <input> tag with all attributes)
  - renderTableRow (returns an HTML <tr> with columns for label and <input>)

PHP code for **`Form_Input`** class

```php
<?php
class Form_Input {
    private $name;
    protected $type;
    public $value = '';
    public $label = '';
    protected $isOutputOnly = false; // boolean

    public function __construct($name, $type='text') {
        $this->name = $name;
        $this->type = $type;
    }

    public function setType( $type ) {
        $this->type = $type;
    }

    public function setOutputOnly() {
        $this->isOutputOnly = true;
    }
```

`Form_Input` class (cont'd.)

```
    public function render() {
        $html = "<input type='{$this->type}'
                        name='{$this->name}'
                        value='{$this->value}' ";
        if ($this->isOutputOnly) {
            $html .= ' disabled="disabled" '
                   . ' class="output-only" ';
        }
        $html .= ' />';
        return $html;
    }
    public function renderTableRow() {
        $html =
            "<tr>
                <td align='right' style='vertical-align:top'>
                    {$this->label}  
                </td>
                <td align='left' style='vertical-align:top'>
                    {$this->render()}
                </td>
            </tr>";
        return $html;
    }
}
```

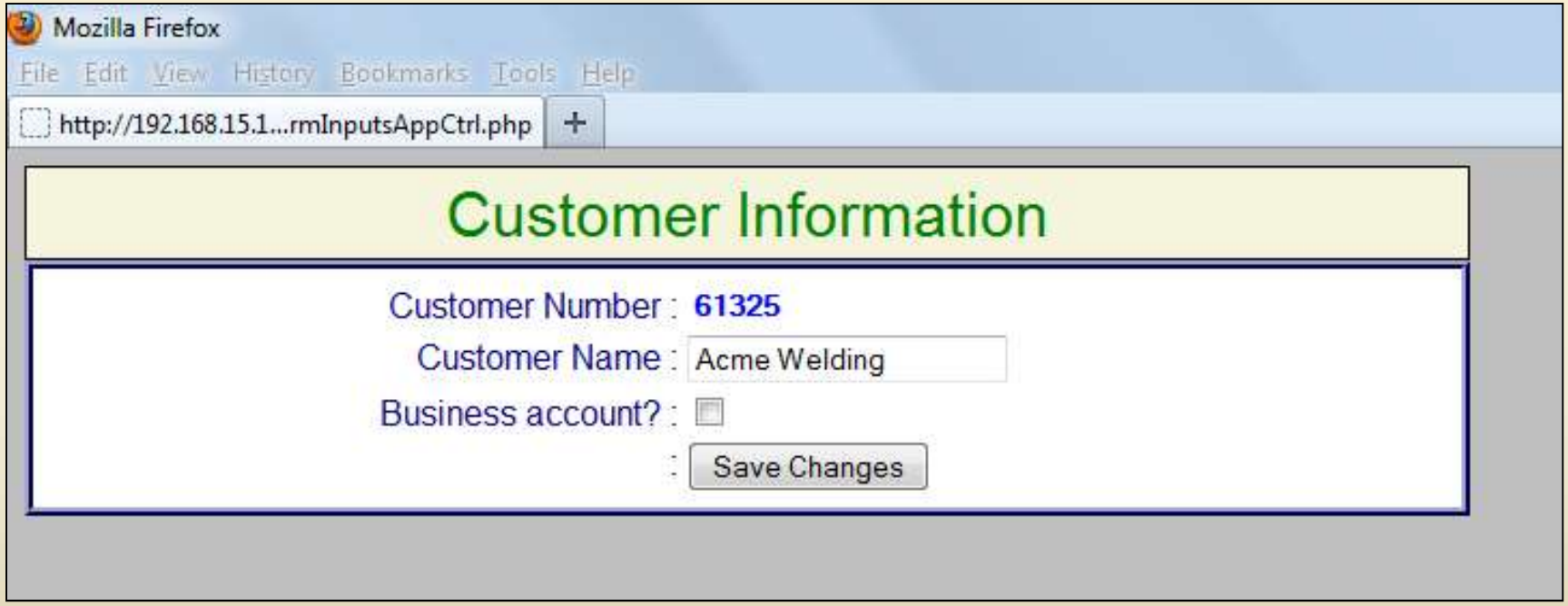
Application using `Form_Input` - *PHP*

```php
require_once 'Form_Input.php';

$inpCustNumber = new Form_Input('CUNUM');
$inpCustNumber->label = 'Customer Number';
$inpCustNumber->value = 61325;
$inpCustNumber->setOutputOnly();

$inpCustName = new Form_Input('CUNAME');
$inpCustName->label = 'Customer Name';
$inpCustName->value = 'Acme Welding';

$inpIsBizCust = new Form_Input('CUBUSINESS', 'checkbox');
$inpIsBizCust->label = 'Business account?';

// Page to return to after form processing
$caller = new Form_Input('caller', 'hidden');
$caller->value = $_SERVER['HTTP_REFERER'];

$submit = new Form_Input('saveButton', 'submit');
$submit->value = 'Save Changes';
```

Application using **`Form_Input`** - *HTML*

```
<html>
    <head>
            <link rel="stylesheet" type="text/css" href="styles.css" />
    </head>
    <body>
        <form>
            <?php echo $caller->render(); ?>
            <table border=0 width="50%">
                <caption>Customer Information</caption>
                <?php
                echo $inpCustNumber->renderTableRow();
                echo $inpCustName->renderTableRow();
                echo $inpIsBizCust->renderTableRow();
                echo $submit->renderTableRow();
                ?>
            </table>
        </form>
    </body>
</html>
```

Customer Input Form – Rendered in FF



IBM i Toolkit

- Toolkit has two components
  - XML Service
    - Created by IBM to provision IBM i resources for other platform development
    - Written in RPG, CL and DB2 stored procedures
  - PHP classes that “wrap” payload of XML Service
    - Built on an Object Oriented model of PHP
    - Use of PHP objects does not require OO knowledge (black box)

Using the IBM i Toolkit

- Need to include the source code (i.e., class definitions) for the toolkit
- Two class files exist:
  - ToolkitService.php
    - Run CL commands
    - Call IBM i programs (RPG, CL, etc. – any *PGM object)
  - iToolkitService.php
    - Access to native IBM i objects
    - Spool files, Data queues, User spaces, System Values, Job Logs, Object Lists
- Use require_once to access Toolkit Classes:
    - `require_once 'ToolkitService.php';`
    - `require_once 'iToolkitService.php';`
- Documentation is in the Zend Server for IBM i User Guide:

  http://files.zend.com/help/Zend-Server-6-IBMi/zend-server.htm#php_toolkit_xml_service_functions.htm

Running CL Commands from PHP

- Create the toolkit object using the singleton design pattern

```php
require_once 'ToolkitService.php';

try {
    $tkObj = ToolkitService::getInstance('*LOCAL', 'PHPUSER', 'phppswd1');
} catch (Exception $e) {
    echo  $e->getMessage(), "\n";
    exit();
}

// Send a message
$cmd = "SNDMSG TOUSR(JVALANCE) MSG('Hi there, John!')";
$tkObj->CLCommand($cmd);

// Change job logging
$cmd = "CHGJOB LOG(4 00 *SECLVL) LOGCLPGM(*YES)";
$tkObj->CLCommand($cmd);
```

- In ToolkitService class, the getInstance method returns an object
  - *Cannot use* `new ToolkitService` *with singleton class*

Examples:
CLInteractiveCommand() and ClCommandWithOutput()

- CLInteractiveCommand:

```
$rows = $tkObj->CLInteractiveCommand("DSPLIBL");
if(!is_array($rows))
    echo $tkObj->getLastError();
else {
    echo "<h2>DSPLIBL output:</h2>";
    foreach ($rows as $row) {
        echo "$row <br>";
    }
}
```

- CLCommandWithOutput:

```
$joba = $tkObj->ClCommandWithOutput("RTVJOBA job(?) user(?) nbr(?)");
if(!$joba) {
    echo $tkObj->getLastError();
} else {
    echo "Job is: {$joba['job']}/{$joba['user']}/{$joba['nbr']}<br/>";
}
```

Thanks for attending!

SUMMARY...

Summary

- OO = encapsulation of data and related functionality
- Class definitions
  - templates for instantiating objects,
  - each object has its own data space.
- Object Instantiation
  - new keyword
- Object member access operator (->)
- $this : access internal members within a class
- Applications: bring class definitions in using require_once()

Summary (cont'd.)

- Member visibility - public and private
  - public is default
  - private is better
- Getter and Setter methods
  - control access to object's data
- Constructor method (__construct)
  - object initialization
- Exception handling,
  - PHP's built-in class: Exception
  - throw new Exception
  - try / catch

More Information

- iPro Developer article on OO PHP by John Valance:
  - http://tinyurl.com/oophp-JV1
  - **April 2013 issue**
- PHP.net:
  - http://php.net/manual/en/language.oop5.php
  - www.php.net/manual/en/class.exception.php
- Contact John Valance :
  - johnv@div1sys.com
  - 802-355-4024
  - http://www.div1sys.com