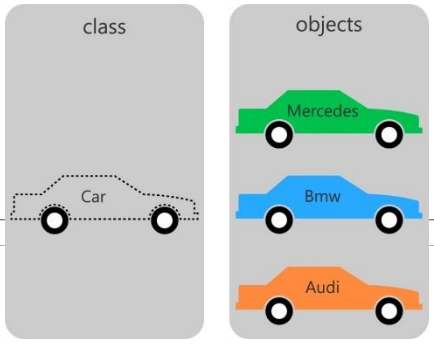
COMP 2021

Unix and Script Programming

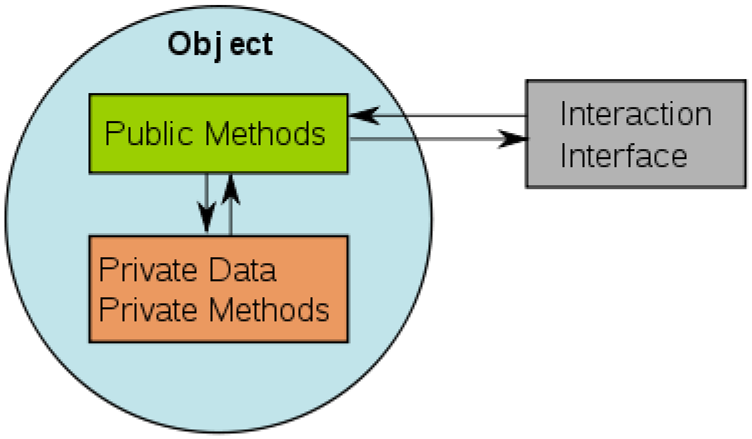


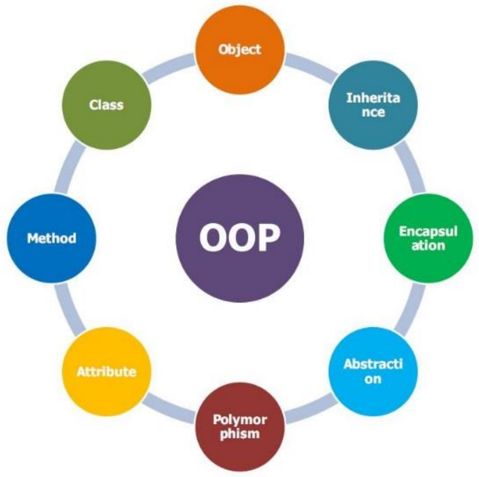
Basic
Object-oriented PHP

Object-Oriented Programming (OOP)

- OOP is a programming paradigm (a style of coding)
- Allows developers to group similar tasks into classes

- Follow 'don't repeat yourself (DRY)' tenet
- Minimum change in code if task is updated





Why Classes and Objects?

- PHP is a primarily procedural language
- Small programs are easily written without adding any classes or objects
- Larger programs, however, become cluttered with so many disorganized functions
- Grouping related data and behavior into objects helps manage size and complexity
- The concept applies to many other programming languages, e.g. C++, Java, Python, etc.

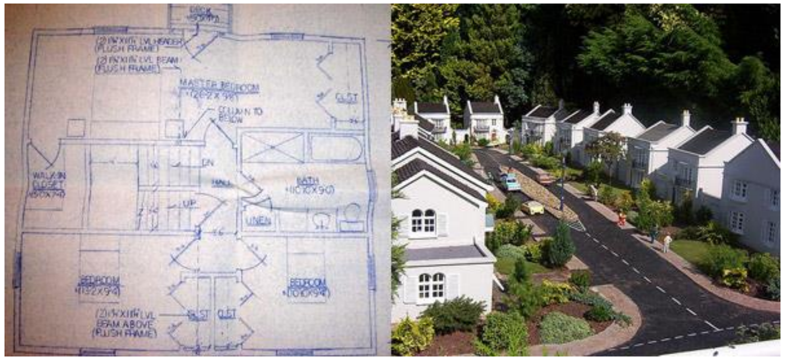
Objected Oriented Concepts

- Data abstraction and encapsulation
- Object, class and instance
- Member variable and member function
- Constructor and destructor
- Inheritance, parent class and child class
- Polymorphism and overloading

Object, Class, Instance

- Object
  - A self-contained component
  - With properties and methods
  - Make a certain type of data useful
- Class
  - A blueprint or template or set of instructions to build a specific type of object.
  - Every object is built from a class.
  - Each class should be designed and programmed to accomplish one, and only one thing
  - Typically, many classes are used to build an entire application
- Instance
  - A specific object built from a specific class

Metaphor: Building a House



- Class: blueprint for a house
  - Blueprint itself is not a house
  - Follow blueprint (instantiate) to make an actual house
- Object: house built according to the blue print
  - Data / Property: wood, wires and concrete that compose the house
  - Method: data needs to be assembled according to blueprint, otherwise it's just a pile of stuff
- Instance: a specific actual house
- Classes form the structure of data and actions and use that information to build objects

Example: Constructing and Using Object (with existing Class)

- Instantiation: make a new instance and is typically done using the `new` keyword
- Test whether a class is installed with `class_exists()`

```php
# construct an object
$name = new ClassName(parameters);
# access an object's field (if the field is public)
$name->fieldName
# call an object's method
$name->methodName(parameters);
```

PHP

```php
<?php
# zip.php unzip a zip file
# use ZipArchive class http://www.php.net/zip
    $zip = new ZipArchive();
    $zip->open("zipExample.zip");
    $zip->extractTo("zipExample/");
    $zip->close();
?>?>
```

PHP

Class Declaration Syntax

```
class ClassName {
  # fields - data inside each object
  public $name; # public field
  private $name; # private field
  # constructor - initializes each object's state
  public function __construct(parameters) {
            statement(s);
  }
  # method - behavior of each object
  public function name(parameters) {
            statements;
  }
}                                                    PHP
```

First OOP PHP Script

```php
<?php
# ooHelloWorld.php
class MyClass{
    public $prop1 = "I'm a class property!";
}
#
$obj = new MyClass;
# see the contents of the class
var_dump($obj);
# access an object
echo $obj->prop1; # Output the property
?>
                                                PHP
```

- –> is an OOP construct that accesses the contained properties and methods of a given object.

Define Class Methods

```php
<?php
# ooClassMethods.php
class MyClass{
    public $prop1 = "I'm a class property!";
    public function setProperty($newval){
        $this->prop1 = $newval;
    }
    public function getProperty(){
        return $this->prop1 . "<br />";
    }
}

$obj = new MyClass;
echo $obj->getProperty(); # Get the property value
$obj->setProperty("I'm a new property value!"); # Set a new one
echo $obj->getProperty(); # Read it out again to show the change
?>                                                             PHP
```

- Objects refer to themselves using `$this` in class declaration

Multiple Instances of the Same Class

- OOP keeps objects as separate entities
- The power of OOP becomes apparent when using multiple instances of the same class

```php
# oo2instances.php
# Create two objects
$obj = new MyClass;
$obj2 = new MyClass;

# Get the value of $prop1 from both objects
echo $obj->getProperty();
echo $obj2->getProperty();

# Set new values for both objects
$obj->setProperty("I'm a new property value!");
$obj2->setProperty("I belong to the second instance!");

# Output both objects' $prop1 value
echo $obj->getProperty();
echo $obj2->getProperty();
                                                                  PHP
```

Constructor

```
<?php
# ooConstructor.php
class MyClass{
  public $prop1 = "I'm a class property!";
  public function __construct(){
        echo 'The class "', __CLASS__, '" was initiated!<br />';
  }
  public function setProperty($newval) {$this->prop1 = $newval;}
  public function getProperty(){return $this->prop1 . "\n";}
}

$obj = new MyClass;
echo $obj->getProperty();
?>
                                                                  PHP
```

- Constructor method `__construct()` is called automatically whenever a new object is created
- It takes care of initialization when an object is instantiated
- `__CLASS__` is a predefined magic constant

Destructor

```php
<?php
# ooDestructor.php
class MyClass {
    public $prop1 = "I'm a class property!";
    public function __construct() {
        echo 'The class "', __CLASS__, '" was initiated!<br />';
    }
    public function __destruct() {
        echo 'The class "', __CLASS__, '" was destroyed.<br />';
    }
    public function setProperty($newval) {$this->prop1 = $newval; }
    public function getProperty() {return $this->prop1 . "<br />"; }
}
$obj = new MyClass;
echo $obj->getProperty();
echo "End of file.<br />";
?>                                                                  PHP
```

- When the end of a file is reached, PHP automatically releases all resources
- The destructor method `__destruct()` is called when the object is destroyed.
  - It is useful for class cleanup (e.g. closing a database connection)
- To explicitly trigger the destructor, you can destroy the object using `unset()`

Magic Methods in PHP

- Magic methods allow to define the reaction when certain event happen to the object
- PHP reserves all function names starting with __ as magical
- Example of the events
  - Construct and destruct: `__construct()`, `__destruct()`
  - Getting and setting: `__get()`, `__set()`
  - Check if set, unset: `__isset()`, `__unset()`
  - Treat the object as a string: `__toString()`
  - Sleep and wakeup: `__sleep()`, `__wakeup()`
  - and more …

Magic Method Example: __toString()

```
<?php
#ooToString.php
class MyClass {
    public $prop1 = "I'm a class property!";
    public function __construct() {
        echo 'The class "', __CLASS__, '" was initiated!<br />';}
    public function __destruct() {
        echo 'The class "', __CLASS__, '" was destroyed.<br />';}
    public function __toString(){
        echo "Using the toString method: ";
        return $this->getProperty();}
    public function setProperty($newval) {
        $this->prop1 = $newval; }
    public function getProperty(){
        return $this->prop1 . "<br />";}
}

$obj = new MyClass;
echo $obj; # treat the object as a string
unset($obj);
?>
```

PHP

Visibility of Properties and Methods

- Methods and properties are assigned visibility for added control over objects
  - Visibility is a new feature as of PHP 5
- Public: accessible anywhere, both within the class and externally
- Protected: accessible within the class itself or in descendant classes
- Private: accessible only from within the class that defines it

Example: Private Method

```php
<?php
# ooPrivate.php
class MyClass{
    public $prop1 = "I'm a class property!";
    public function __construct(){
        echo 'The class "', __CLASS__, '" was initiated!<br />';}
    public function __destruct(){
        echo 'The class "', __CLASS__, '" was destroyed.<br />';}
    public function __toString() {
        echo "Using the toString method: ";
        return $this->getProperty();
    }
    public function setProperty($newval) {
        $this->prop1 = $newval;}
    private function getProperty(){
        return $this->prop1 . "<br />";}
}

$newobj = new MyClass;
# fatal error: Call to private method MyClass::getProperty()
echo $newobj->getProperty();
?>
```

Class Inheritance

- Classes can inherit the methods and properties of another class using the `extends` keyword.

```
<?php
# ooInheritance.php
class MyClass {
    public $prop1 = "I'm a class property!";
    public function __construct(){
        echo 'The class "', __CLASS__, '" was initiated!<br />'; }
    public function __destruct(){
        echo 'The class "', __CLASS__, '" was destroyed.<br />'; }
    public function __toString() {
        return $this->getProperty(); }
    public function setProperty($newval) {
        $this->prop1 = $newval; }
    public function getProperty() {
        return $this->prop1 . "<br />"; }
}

class MyOtherClass extends MyClass {
    public function newMethod() {
        echo "From a new method in " . __CLASS__ . ".<br />"; }
}

$newobj = new MyOtherClass;
echo $newobj->newMethod();
echo $newobj->getProperty();
?>
```

Overwriting Inherited Properties and Methods

- To change the behavior of an existing property or method in the new class, simply overwrite it by declaring it again in the new class

```
class MyOtherClass extends MyClass
{
    #overwrite the constructor in MyClass
    public function __construct(){
        echo "A new constructor in " . __CLASS__ . ".<br />";
    }

    public function newMethod(){
        echo "From a new method in " . __CLASS__ . ".<br />";
    }
}
# refer to ooOverwrite.php for full script
```

Preservation while Overloading

- How to add new functionality to an inherited method while keeping the original method intact?
- Use the parent keyword with the scope resolution operator (: :)

```
class MyOtherClass extends MyClass {
  public function __construct() {
      # call constructor from parent class
      parent::__construct();
      echo "A new constructor in " . __CLASS__ . ".<br
/>";}

# refer to ooScopeResolution.php for full script
```

DocBlocks

- The DocBlock commenting style is a widely accepted method of documenting classes
  - Block comment starts with an additional *
  - Powerful with ability to use tags: `@author`, `@copyright`, `@license`, `@var`, `@param`, `@return`

Example: DocBlock

```
<?php
#Full example at ooDocBlock.php

/**
 * A simple class
 *
 * This is the long description for this class.
 * It may span as many lines as needed.
 * Not compusorly but nice to have.
 *
 * It can also span multiple paragraphs.
 *
 * @author Cindy LI <lixin@cse.ust.hk>
 * @copyright 2016 Cindy LI
 * @license http://www.php.net/license/3_01.txt PHP License 3.01
 */

class SimpleClass {
    /**
     * A public variable
     *
     * @var string stores data for the class
     */
    public $foo;
```

PHP