**PHP Object Oriented Programming (OOPs) concept Tutorial with Example**

**What is OOPs?**

Object Oriented is an approach to software development that models application around real world objects such as employees, cars, bank accounts, etc. A class defines the properties and methods of a real world object. An object is an occurrence of a class.

The three basic components of object orientation are;

* Object oriented analysis – functionality of the system
* Object oriented designing – architecture of the system
* Object oriented programming – implementation of the application

**Object Oriented Programming Principles**

The three major principles of OOP are;

* **Encapsulation** – this is concerned with hiding the implementation details and only exposing the methods. The main purpose of encapsulation is to;
  + Reduce software development complexity – by hiding the implementation details and only exposing the operations, using a class becomes easy.
  + Protect the internal state of an object – access to the class variables is via methods such as get and set, this makes the class flexible and easy to maintain.
  + The internal implementation of the class can be changed without worrying about breaking the code that uses the class.
* **Inheritance**– this is concerned with the relationship between classes. The relationship takes the form of a parent and child. The child uses the methods defined in the parent class. The main purpose of inheritance is;
  + Re-usability– a number of children, can inherit from the same parent. This is very useful when we have to provide common functionality such as adding, updating and deleting data from the database.
* **Polymorphism** – this is concerned with having a single form but many different implementation ways. The main purpose of polymorphism is;
  + Simplify maintaining applications and making them more extendable.

**OOPs Concepts in PHP**

PHP is an object oriented scripting language; it supports all of the above principles. The above principles are achieved via;

* **Encapsulation** - via the use of “get” and “set” methods etc.
* **Inheritance** - via the use of extends keyword
* **Polymorphism** - via the use of implements keyword

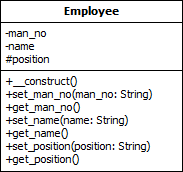
Now that we have the basic knowledge of OOP and how it is supported in PHP, let us look at examples that implement the above principles

**What is UML?**

Unified Modeling Language UML is a technique used to design and document object oriented systems.

UML produces a number of documents, but we will look at the class diagram which is very important to object oriented php programming.

**Class Diagram Example**

[](https://www.guru99.com/images/2013/04/employee_class.png)

**Class Diagram Key**

* The **Upper box** contains the class name
* The **middle box** contains the class variables
* The **lower box** contains the class methods
* The **minus (-)** sign meansprivate scope
* The **plus (+)** sign means public scope
* The **hash (#)** sign means protected scope

**How to Create a class in PHP**

The class keyword is used to define a class in PHP. Below are the rules for creating a class in PHP.

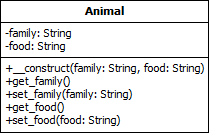
* The class name should start with a letter
* The class name cannot be a PHP reserved word
* The class name cannot contain spaces

Let’s say we want to create a class for representing animals.

We will start with identifying the features that are common to all animals.

* All animals belong to a family such as a herbivore, carnival, etc.
* All animals eat food

The diagram below shows the diagram for the animal

[](https://www.guru99.com/images/2013/04/animal_class.png)

Let’s now code our animal class

<?php

class Animal

{

private $family;

private $food;

public function \_\_construct($family, $food)

{

$this->family = $family;

$this->food = $food;

}

public function get\_family()

{

return $this->family;

}

public function set\_family($family)

{

$this->family = $family;

}

public function get\_food()

{

return $this->food;

}

public function set\_food($food)

{

$this->food = $food;

}

}

?>

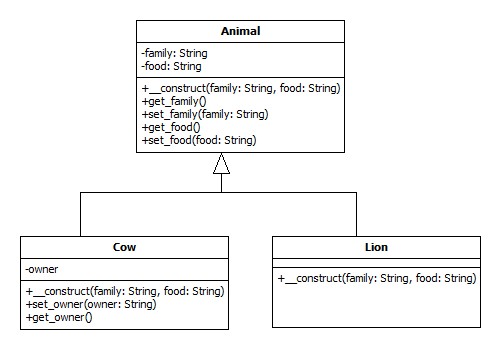
  HERE,

* “private $family, $food” means the variables cannot be accessed directly outside the class (Encapsulation).
* “public function \_\_construct($family…)” is the php constructor method. This function is called whenever an instance of the class has been created. In this case, we are setting the family and food.
* “public function get…()” is the method used to access the family or food value (Encapsulation)
* “public function set…()” is the method used to set the family or food value (Encapsulation)

**How implement Inheritance in PHP**

We will work with a cow and a lion. Both the cow and lion inherit from the Animal class.

The class diagram below shows the relationships.

[](https://www.guru99.com/images/2013/04/inheritance.jpg)

Note the cow inherits from the animal class and defines its own variable and methods too.

Let’s now code the Cow class

<?php

class Cow extends Animal

{

private $owner;

public function \_\_construct($family, $food)

{

parent::\_\_construct($family, $food);

}

public function set\_owner($owner)

{

$this->owner = $owner;

}

public function get\_owner()

{

return $this->owner;

}

}

?>

  Let’s now code the Lion class

<?php

class Lion extends Animal

{

public function \_\_construct($family, $food)

{

parent::\_\_construct($family, $food);

}

}

?>

  HERE,

* “class … extends Animal” makes the cow and lion use methods from the Animal class (Inheritance).

**How to Create object of the class**

The Animal, Cow, and Lion classes should all be in the same directory for simplicity’s sake.

Let’s now create the application that uses our classes.

PHP Class Example

<?php

require 'Animal.php';

require 'Cow.php';

require 'Lion.php';

$cow = new Cow('Herbivore', 'Grass');

$lion = new Lion('Canirval', 'Meat');

echo '<b>Cow Object</b> <br>';

echo 'The Cow belongs to the ' . $cow->get\_family() . ' family and eats ' . $cow->get\_food() . '<br><br>';

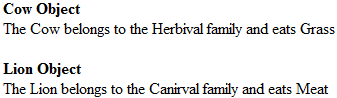
echo '<b>Lion Object</b> <br>';

echo 'The Lion belongs to the ' . $lion->get\_family() . ' family and eats ' . $lion->get\_food();

?>

**Testing our application**

Let’s now view our application in a web browser

. [](https://www.guru99.com/images/2013/04/animal_object.png)

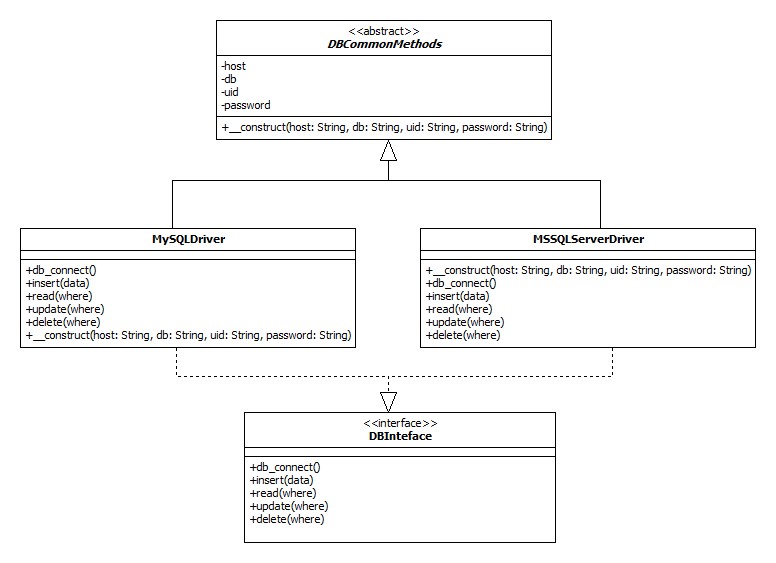
Fantastic right! Let’s now look at the third principle of OOP, polymorphism.

Let’s say we want to develop an application that connects to different database engines such as MySQL and[SQL](https://www.guru99.com/sql.html)Server but use the same uniform interface.

We can create an interface that defines the standard methods and an abstract class that implements the common methods.

* **Interface** – it is similar to a class. It only defines the methods and parameters.
* **Abstract class**– it is a class that cannot be used to create an object directly. Its purpose is to provide partial or whole implementations of common methods.

The class diagram below illustrates the relationship among our abstract class, interface, and implementation classes.

[](https://www.guru99.com/images/2013/04/inheritance2.jpg)

Let’s now create our abstract class

<?php

abstract class DBCommonMethods

{

private $host;

private $db;

private $uid;

private $password;

public function \_\_construct($host, $db, $uid, $password)

{

$this->host = $host;

$this->db = $db;

$this->uid = $uid;

$this->password = $password;

}

}

?>

  HERE,

* “abstract class” means the class cannot be used directly to php create object
* “$host,$db…” are class variables common to all implementations
* “function \_\_construct(…)” is the php class constructor method that sets the common variables values at initialization

Let’s now create the interface that contains the standard methods which will be implemented differently depending on the database engine.

<?php

interface DBInterface

{

public function db\_connect();

public function insert($data);

public function read($where);

public function update($where);

public function delete($where);

}

?>

  HERE,

* “interface” is the keyword for creating interfaces
* “public function…(…)” are the standard methods that should be implemented

Let’s now create the concrete classes that will extend the DBCommonMethods class and extend the DBInterface interface. MySQLDriver.php

<?php class MySQLDriver extends

DBCommonMethods implements DBInterface { public function \_\_construct($host, $db, $uid, $password)

{

parent::\_\_construct($host, $db, $uid, $password); }

public function db\_connect() { //connect code goes here }

public function delete($where) { //delete code goes here }

public function insert($data) { //insert code goes here }

public function read($where) { //read code goes here }

public function update($where) { //update code goes here }

} ?>

  MSSQLServerDriver.php

<?php

class MSSQLServerDriver extends

DBCommonMethods implements DBInterface { public function \_\_construct($host, $db, $uid, $password)

{

parent::\_\_construct($host, $db, $uid, $password); }

public function db\_connect() { //connect code goes here }

public function delete($where) { //delete code goes here }

public function insert($data) { //insert code goes here }

public function read($where) { //read code goes here }

public function update($where) { //update code goes here }

} ?>

  HERE,

* “class … extends DBCommonMethods” use the methods in the DBCommonMethods
* “… implements DBInterface” ensures that the class provides standard methods regardless of the database driver used.

Usage of the above code The code using the above class would look like this

<?php $db = new MySQLDriver($host,$db,$uid,$password); ?>

Or

<?php $db = new MSSQLServerDriver ($host,$db,$uid,$password); ?>

The rest of the code would be the same for both drivers such as;

<?php

$db->db\_connect();

$db->insert($data);

?>

**Summary**

* Object Oriented Programming OOP is a powerful technical that models applications after real world objects
* A class is a representation of real world objects with properties and methods
* The three basic principles of OOP are;
  + Encapsulation
  + Inheritance
  + Polymorphism