Object-Oriented PHP With Classes and Objects .

What Is Object-Oriented Programming (OOP)?

An approach which helps you to develop complex applications in a way that's easily maintainable and scalable over the long term.

What Is a PHP Class?

A class is a template which represents a real-world entity, and it defines properties and methods of the entity.

```php
<?php
class Employee {
  private $first_name;
  private $last_name;
  private $age;
  public function __construct($first_name, $last_name, $age){
    $this->first_name = $first_name;
    $this->last_name = $last_name;
    $this->age = $age;
}
  public function getFirstName(){
    return $this->first_name;
  }
  public function getLastName(){
    return $this->last_name;
  }
  public function getAge(){
    return $this->age;
  }
}
?>
```

Class Properties in PHP

You could think of class properties as variables that are used to hold information about the object. In the above example, we've defined three properties `first_name`, `last_name`, and `age`. In most cases, class properties are accessed via instantiated objects.

Constructors for PHP Classes

A constructor is a special class method which is called automatically when you instantiate an object.

What Is an Object in PHP?

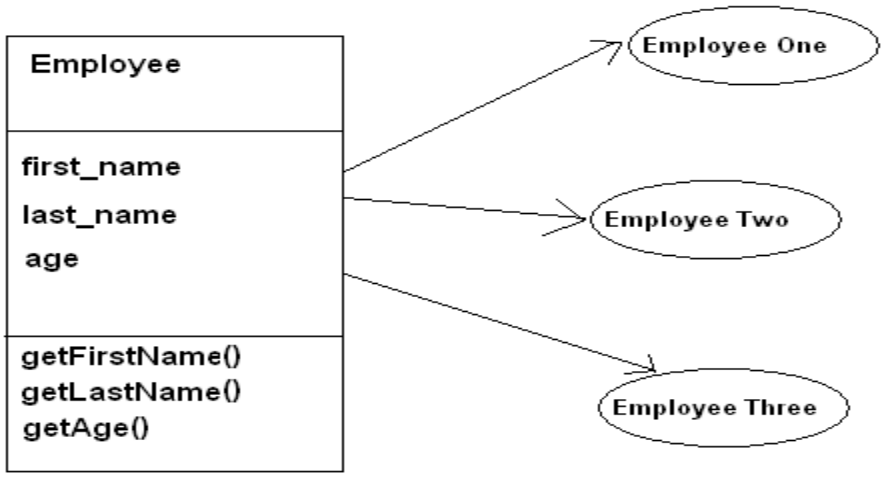
```
<?php
$objEmployee = new Employee('Bob', 'Smith', 30);
echo $objEmployee->getFirstName(); // print 'Bob'
echo $objEmployee->getLastName(); // prints 'Smith'
echo $objEmployee->getAge(); // prints '30'
?>
```

You need to use the `new` keyword when you want to instantiate an object of any class along with its class name, and you'll get back a new object instance of that class.

If a class has defined the `__construct` method and it requires arguments, you need to pass those arguments when you instantiate an object. In our case, the `Employee` class constructor requires three arguments, and thus we've passed these when we created the `$objEmployee` object. As we discussed earlier, the `__construct` method is called automatically when the object is instantiated.

Next, we've called class methods on the `$objEmployee` object to print the information which was initialized during object creation. Of course, you can create multiple objects of the same class, as shown in the following snippet.

```php
<?php
$objEmployeeOne = new Employee('Bob', 'Smith', 30);
echo $objEmployeeOne->getFirstName(); // prints 'Bob'
echo $objEmployeeOne->getLastName(); // prints 'Smith'
echo $objEmployeeOne->getAge(); // prints '30'
$objEmployeeTwo = new Employee('John', 'Smith', 34);
echo $objEmployeeTwo->getFirstName(); // prints 'John'
echo $objEmployeeTwo->getLastName(); // prints 'Smith'
echo $objEmployeeTwo->getAge(); // prints '34'
?>
```



Encapsulation

Encapsulation is an important aspect of OOP that allows you to restrict access to certain properties or methods of the object. And that brings us to another topic for discussion: access levels.

Access Levels : When you define a property or a method in a class, you can declare it to have one of these three access levels—`public`, `private`, or `protected`.

Public Access : When you declare a property or a method as public, it can be accessed from anywhere outside the class. The value of a public property can be modified from anywhere in your code.

```php
<?php
class Person
{
  public $name;
  public function getName()
  {
    return $this->name;
  }
}
$person = new Person();
$person->name = 'Bob Smith';
echo $person->getName(); // prints 'Bob Smith'
?>
```

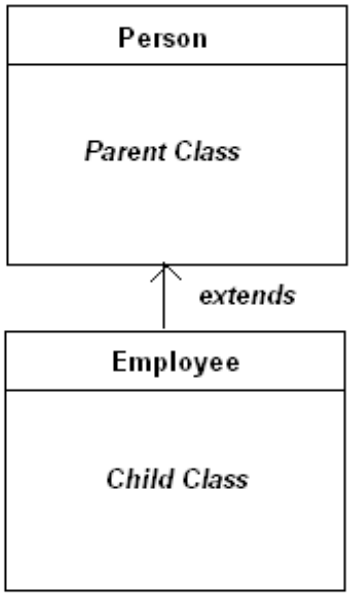
Private Access : *When you declare a property or a method as* `private`*, it can only be accessed from within the class. This means that you need to define getter and setter methods to get and set the value of that property.*

```php
<?php
class Person
{
  private $name;
  public function getName()
  {
    return $this->name;
  }
  public function setName($name)
  {
    $this->name = $name;
  }
}
$person = new Person();
$person->name = 'Bob Smith'; // Throws an error
$person->setName('Bob Smith');
echo $person->getName(); // prints 'Bob Smith'
?>
```

Protected Access : *when you declare a property or a method as* `protected`*, it can be accessed by the same class that has defined it and classes that inherit the class in question. We'll discuss inheritance in the very next section, so we'll get back to the protected access level a bit later.*

Inheritance

Inheritance is an important aspect of the object-oriented programming paradigm which allows you to inherit properties and methods of other classes by extending them. The class which is being inherited is called the **parent class**, and the class which inherits the other class is called the **child class**. When you instantiate an object of the child class, it inherits the properties and methods of the parent class as well.



```php
<?php
class Person
{
 protected $name;
  protected $age;
  public function getName()
  {
    return $this->name;
  }
  public function setName($name)
  {
    $this->name = $name;
  }
  private function callToPrivateNameAndAge()
  {
    return "{$this->name} is {$this->age} years old.";
  }
  protected function callToProtectedNameAndAge()
  {
    return "{$this->name} is {$this->age} years old.";
  }
}
```

```php
class Employee extends Person
{
  private $designation;
  private $salary;
  public function getAge()
  { return $this->age; }
  public function setAge($age)
  { $this->age = $age; }

  public function getDesignation()
  { return $this->designation; }
  public function setDesignation($designation)
  { $this->designation = $designation;}

  public function getSalary()
  { return $this->salary; }
  public function setSalary($salary)
  { $this->salary = $salary; }
  public function getNameAndAge()
  { return $this->callToProtectedNameAndAge(); }
}
```

```php
$employee = new Employee();
$employee->setName('Bob Smith');
$employee->setAge(30);
$employee->setDesignation('Software Engineer');
$employee->setSalary('30K');
echo $employee->getName(); // prints 'Bob Smith'
echo $employee->getAge(); // prints '30'
echo $employee->getDesignation(); // prints 'Software Engineer'
echo $employee->getSalary(); // prints '30K'
echo $employee->getNameAndAge(); // prints 'Bob Smith is 30 years old.'
echo $employee->callToPrivateNameAndAge(); // produces 'Fatal Error'
?>
```

The important thing to note here is that the `Employee` class has used the `extends` keyword to inherit the `Person` class. Now, the `Employee` class can access all properties and methods of the `Person` class that are declared as public or protected. (It can't access members that are declared as private.)

In the above example, the `$employee` object can access `getName` and `setName` methods that are defined in the `Person` class since they are declared as public.

Next, we've accessed the `callToProtectedNameAndAge` method using the `getNameAndAge` method defined in the `Employee` class, since it's declared as protected. Finally, the `$employee` object can't access the `callToPrivateNameAndAge` method of the `Person` class since it's declared as private.

On the other hand, you can use the `$employee` object to set the `age` property of the `Person` class, as we did in the `setAge` method which is defined in the `Employee` class, since the `age` property is declared as protected.

So that was a brief introduction to inheritance. It helps you to reduce code duplication, and thus encourages code reusability.

Polymorphism

Polymorphism is another important concept in the world of object-oriented programming which refers to the ability to process objects differently based on their data types.

For example, in the context of inheritance, if the child class wants to change the behavior of the parent class method, it can override that method. This is called method overriding. Let's quickly go through a real-world example to understand the concept of method overriding.

```php
<?php
class Message
{
  public function formatMessage($message)
  {
    return printf("<i>%s</i>", $message);
  }
}
class BoldMessage extends Message
{
  public function formatMessage($message)
  {
    return printf("<b>%s</b>", $message);
  }
}
$message = new Message();
$message->formatMessage('Hello World'); // prints '<i>Hello World</i>'
$message = new BoldMessage();
$message->formatMessage('Hello World'); // prints '<b>Hello World</b>'
?>
```

As you can see, we've changed the behavior of the `formatMessage` method by overriding it in the `BoldMessage` class. The important thing is that a message is formatted differently based on the object type, whether it's an instance of the parent class or the child class.