Поговорим о:

Объектно-ориентированном программировании

ООП

Объектно-ориентированное программирование — это стиль кодирования, который позволяет разработчику группировать схожие задачи в **классы**.
Стратегию ООП - смещение приоритетов в процессе программирования от функциональности приложения к структурам данных. Это позволяет программисту моделировать в создаваемых приложениях реальные объекты и ситуации.
Технология ООП обладает тремя главными преимуществами:

- она проста для понимания: ООП позволяет мыслить категориями повседневных объектов;
- повышено надежна и проста для сопровождения — правильное проектирование обеспечивает простоту расширения и модификации объектно-ориентированных программ. Модульная структура позволяет вносить независимые изменения в разные части программы, сводя к минимуму риск ошибок программирования;
- ускоряет цикл разработки — модульность и здесь играет важную роль, поскольку различные компоненты объектно-ориентированных программ можно легко использовать в других программах, что уменьшает избыточность кода и снижает риск внесения ошибок при копировании.

Объектно-ориентированное программирование основано на:

- Инкапсуляции;
- Полиморфизме;
- Наследовании.

ИНКАПСУЛЯЦИЯ

Инкапсуляция - это механизм, объединяющий данные и обрабатывающий их код как единое целое.

Многие преимущества ООП обусловлены одним из его фундаментальных принципов — **инкапсуляцией**.
Инкапсуляцией называется включение различных мелких элементов в более крупный объект, в результате чего программист работает непосредственно с этим объектом. Это приводит к упрощению программы, поскольку из нее исключаются второстепенные детали.

Инкапсуляцию можно сравнить с работой автомобиля с точки зрения типичного водителя. Многие водители не разбираются в подробностях внутреннего устройства машины, но при этом управляют ею именно так, как было задумано. Пусть они не знают, как устроен двигатель, тормоз или рулевое управление, — существует специальный интерфейс, который автоматизирует и упрощает эти сложные операции. Сказанное также относится к инкапсуляции и ООП — многие подробности "внутреннего устройства" скрываются от пользователя, что позволяет ему сосредоточиться на решении конкретных задач. В ООП эта возможность обеспечивается классами, объектами и различными средствами выражения иерархических связей между ними.

НАСЛЕДОВАНИЕ

Наследование (inheritance) - это процесс, посредством которого один объект может приобретать свойства другого. Точнее, объект может наследовать основные свойства другого объекта и добавлять к ним черты, характерные только для него. Не путайте с копированием объектов. При копировании создается точная копия объекта, а при наследовании точная копия дополняется уникальными свойствами, которые характерны только для производного объекта.Наследование является важным, поскольку оно позволяет поддерживать концепцию иерархии классов (hierarchical classification).

Например, подумайте об описании жилого дома.

Дом - это часть общего класса, называемого **строением.**

С другой стороны, **строение** - это часть более общего класса - **конструкции,** который является частью ещё более общего класса объектов, который можно назвать **созданием рук человека**.

В каждом случае порождённый класс наследует все, связанные с родителем, качества и добавляет к ним свои собственные определяющие характеристики. Без использования иерархии классов, для каждого объекта пришлось бы задать все характеристики, которые бы исчерпывающи его определяли. Однако при использовании наследования можно описать объект путём определения того общего класса, к которому он относится, с теми специальными чертами, которые делают объект уникальным.

ПОЛИМОРФИЗМ

Полиморфизм (многоформенность) является следствием идеи наследования. Собственно полиморфность класса — это свойство базового класса использовать функции производных классов, даже если на момент определения еще неизвестно, какой именно класс будет включать его в качестве базового и, тем самым, становиться от него производным.

Собственно механизм больше связан с переопределением части функций базового класса в наследующем.

```
class Base {
	function funct() {
		echo "<h2>Функция базового класса</h2>";
	}
	function base_funct() {
		$this->funct();
	}
}

class Derivative extends Base {
	function funct() {
		echo "<h3>Функция производного класса</h3>";
	}
}

$b = new Base();
$d = new Derivative();
$b->base_funct();
$d->funct();
$d->base_funct();
// Скрипт выводит:
// Функция базового класса
// Функция производного класса
// Функция производного класса
```

КЛАСЫ И ОБЪЕКТЫ

Класс является основой для объектов. Это блок кода, который определяет:
Типы данных, которые объекты, созданные из класса, будут хранить и функции, которые эти объекты будут содержать.
При создании объектно-ориентированного приложения, вы обычно создаете один или более классов, представляющих различные типы объектов в вашем приложении. Например, если вы пишете сайт, вы можете создать классы, называемые *Site*, *Article*, *Category* и *User*.

Объект — это **специальный тип переменной, который создается из класса**. Он содержит фактические данные, и вы можете вызывать объектные функции, чтобы работать с этими данными. Из одного класса вы можете создать сколько угодно объектов. Каждый объект функционирует независимо от других, даже если все они из одного и того же класса.

Разложим по полочкам:

Класс, например, это проект дома. Он определяет на бумаге как будет выглядеть дом, чётко описывает все взаимосвязи между его различными частями, даже если дом не существует в реальности.
А объект — это реальный дом, который построен в соответствии с проектом. Данные, которые хранятся в объекте похожи на дерево, провода и бетон, из которых построен дом: без сборки в соответствии с проектом, они будут всего лишь кучей материалов . Однако, собранные вместе они становятся отличным и удобным домом.

Классы формируют структуру данных и действий и используют эту информацию для строительства объектов. Из одного класса может быть построено более одного объекта в одно и тоже время, каждый из них будет независим от других. Продолжая аналогию со строительством, целый район может быть построен по одному проекту: 150 различных домов, которые выглядят одинаково, но в каждом из них живут разные семьи и внутренняя отделка строений разная.

СОЗДАНИЕ КЛАССА

Базовый синтаксис для создания класса в PHP выглядит так:

```
<?php
	class ИмяКласса
	{
		// (Здесь находится описание класса)
	}
?>
```

Например, если мы хотим создать класс User на своём веб-сайте:

```
<?php
	class User
	{
		// (Здесь находится описание класса)
	}
?>
```

Довольно просто не правда ли?! Конечно, этот класс ничего не может делать, до тех пор пока вы не добавите **свойства и методы** этого класса. Тем не менее, приведенный выше код создает настоящий класс, пригодный для использования в PHP.

Хорошим тоном в ООП программировании является то, что для каждого класса нужно создавать свой собственный файл, с тем же именем, что и имя класса. Например, вы могли бы записать указанный выше код класса User в файл User.php и сохранить файл в папку classes.

СОЗДАНИЕ ОБЪЕКТОВ

Вы можете создать объект класса с помощью ключевого слова new, следующим образом:

```
<?php
    new Имя_класса()

?>
```

Эта конструкция создает объект на базе класса «Имя_класса» . Обычно, чтобы хранить вновь созданный объект, нужно присвоить его переменной, которую можно использовать позже для доступа к объекту. Например, давайте создадим новый объект User и сохраним его в переменную $User:

```
<?php
        $user = new User();
?>
```

Также мы можем создать еще один объект в этом же классе:

```
<?php

        $user2 = new User();
?>
```

Даже если мы создали эти классы из одного и того же класса User, то эти объекты $User и $User2 абсолютно не зависят друг от друга.

СВОЙСТВА КЛАССА

Свойства, или переменные класса, используются для того, чтобы добавить данные в класс. Свойства работают как обычные переменные, но они связаны с объектом, и к ним можно получить доступ только используя объект.

Существует 3 типа свойств, которые можно добавить в класс:

- Свойства с ключевым словом ***public*** они будут доступны везде. Эти свойства можно прочитать или изменить с помощью любого кода в вашем сценарии, будь этот код внутри или вне класса.
  Собственно после создания объекта можно напрямую обращаться к этим свойствам.

- Свойства с ключевым словом ***private*** доступны только с помощью методов в пределах класса. Если это возможно, то лучше всего делать ваши свойства недоступными извне, с помощью ключевого слова *private*, поскольку это помогает сохранить объекты отделенными от остальной части вашего кода.
  После создания объекта нельзя напрямую обращаться к этим свойствам, с ними работают методы класса.

- Свойства с ключевым словом ***protected*** могут быть доступны с помощью методов внутри класса, а также в классах, которые являются наследниками родительского класса. Чтобы добавить в класс свойство, нужно перед ним написать одно из ключевых слов *public*, *private* или *protected*:

ДОСТУП К СВОЙСТВАМ

```
class MyClass
{
    public $public = 'Общий';
    protected $protected = 'Защищенный';
    private $private = 'Закрытый';

    function printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}

$obj = new MyClass();
echo $obj->public; // Работает
echo $obj->protected; // Неисправимая ошибка
echo $obj->private; // Неисправимая ошибка
$obj->printHello(); // Выводит Общий, Защищенный и Закрытый
```

МЕТОДЫ КЛАССА

Методы - это обычные функции, которые определяются внутри класса, они позволяют объектам выполнять различные задачи.
Объявление метода напоминает определение обычной функции, за исключением предваряемого одного из ключевых слов protected, public, или private. Если в определении метода вы опустите ключевое слово, определяющее видимость, то метод будет объявлен неявно как public.
К методам объекта можно обращаться с помощь символов '->', указав объект и имя метода. При вызове метода, так же как и при вызове функции нужно использовать круглые скобки.

Методы класса должны быть определены через модификаторы public, private, или protected. Методы, где определение модификатора отсутствует, поумолчанию определяются как public.

```php
<?php

 class first {
  public $str = 'some text';

  // определение метода
  function getstr() {
    echo $this->str;
  }
 }

 $obj = new first();

 // вызов метода объекта
 $obj->getstr();

?>
```

ОБЛАСТЬ ВИДИМОСТИ МЕТОДА

```
class MyClass
{
// Объявление общедоступного конструктора
public function __construct() { }

// Объявление общедоступного метода
 function MyPublic() { }

// Объявление защищенного метода
protected function MyProtected() { }

// Объявление закрытого метода
private function MyPrivate() { }

 // Это общедоступный метод
   function Foo()
   {
      $this->MyPublic();
      $this->MyProtected();
      $this->MyPrivate();
   }
}
```

```
$myclass = new MyClass;
$myclass->MyPublic(); // Работает
$myclass->MyProtected(); // Неисправимая ошибка
$myclass->MyPrivate(); // Неисправимая ошибка
$myclass->Foo(); // Работает общий, защищенный и закрыт
ый
```

МАГИЧЕСКИЕ МЕТОДЫ

Любой нативный метод в php, который начинается с __ называются магическим. Вся «магия» таких методов состоит в том, что они могут вызываться при совершении какого-то действия автоматически и без ведома программиста.
Наиболее важными считаются магические методы - **__construct() и __destructor().**

Не менее значимые можно считать и магические методы-перехватчики ***__get, __set, __isset, __unset, __call***.
Назвали их так за то, что они словно перехватывают обращение к недоступным или несуществующим членам класса.

Есть и другие «магические методы», как **__callStatic(), __sleep(), __wakeup(), __toString(), __invoke(), __set_state(), __clone(), __debugInfo()**

Магические методы могут показаться очень полезными, но злоупотреблять их использованием всё же не стоит. Код, который целиком построенный на методах-перехватчиках, сложно поддаётся анализу, ведь методы и поля, которые класс получил при из использовании в нём не объявлены и их наличие может легко ввести в заблуждение. Код больше времени читают и анализируют, нежели пишут и, возможно, вы сэкономите время, потраченное на написание кода, если будете использовать методы-перехватчики, но вы или сторонние разработчики будут анализировать ваш код намного дольше.

__construct()

Когда создаётся объект, очень часто нужно, чтобы при этом сразу производились установки некоторых свойств. Для выполнения таких задач PHP имеет магический метод __construct(), который вызывается автоматически при создании нового объекта.

Примечание — константа **__CLASS__** возвращает имя класса, в котором она вызывается; это одна из магических констант PHP.

```
<?php
class MyClass{
  public $prop1 = "Свойство класса ";
  public function __construct()    {
    echo 'Создан объект класса "'. __CLASS__. '"!<br />';
  }
  public function setProperty($newval)    {
    $this->prop1 = $newval;
  }
  public function getProperty()    {
    return $this->prop1 . "<br />";
  }
}
// Создаем новый объект
$obj = new MyClass;
// Получаем значение свойства $prop1
echo $obj->getProperty();
// Выводим сообщение о достижении конца файла
echo "Конец файла.<br />";
?>
```

__destruct()

Для вызова функции в процессе удаления объекта используется магический метод __destruct().
Это полезный метод для корректной очистки свойств класса (например, для правильного закрытия соединения с БД).

```
class MyClass
{
    public $prop1 = "Свойство класса ";
    public function __construct()    {
        echo 'Создан объект класса "', __CLASS__, '"!<br />';
    }
    public function __destruct()    {
        echo 'Объект класса "', __CLASS__, '" удален.<br />';
    }
    public function setProperty($newval)    {
        $this->prop1 = $newval;
    }
    public function getProperty()    {
        return $this->prop1 . "<br />";
    }
}
```

```
// Создаём новый объект
$obj = new MyClass;

// Получаем значение свойства $prop1
echo $obj->getProperty();

// Выводим сообщение о достижении конца файла
echo "Конец файла.<br />";
```

Для явного вызова деструктора и удаления объекта Вы можете использовать функцию unset():

```
unset($obj);
```

Полезные ссылки:

http://www.studfiles.ru/preview/915306/ - хорошо написано про объектно-ориентированный подход.
http://true-coder.ru/oop-v-php/oop-v-php-magicheskie-metody-metody-perexvatchiki.html - магические методы в PHP.
https://php.ru/manual/language.oop5.magic.html - еще магические методы.