Разработка многостраничного сайта на PHP

ЗАНЯТИЕ 1.
ВВЕДЕНИЕ В ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Тема занятия. Введение в объектно-ориентированное программирование

Цель занятия –

Изучить основы объектно-ориентированного программирования, познакомится с понятиями «класс», «объект», «конструктор класса»

Актуализация

На прошлых занятиях мы обсудили HTTP – заголовки ответа сервера, разобрались с формами в работа с PHP. Теперь нам предстоит изучить основы объектно-ориентированного программирования.

1) Что такое ООП?
2) Что такое классы и объекты и чем они отличаются?
3) Какие есть псевдопеременные?
4) Какие есть конструкторы класса PHP?

Классы и объекты в PHP

Объектно-ориентированное программирование (ООП) - это стиль программирования, предназначенный для того, чтобы приблизить мышление о программировании к мышлению о реальном мире.

Объекты создаются с помощью классов, которые являются ключевым элементом ООП.
Класс описывает, каким будет объект, но отделен от самого объекта. Другими словами, класс можно рассматривать как чертеж, описание или определение объекта.

Взгляните на следующие примеры:

Классы и объекты в PHP

Классы и объекты в PHP

Здесь, **Building** (здание) - это класс. Он определяет особенности общего здания и то, как оно должно работать. **Empire State Building** (Эмпайр-стейт-билдинг) - это конкретный объект (или экземпляр (**instance**)) этого класса.

Вы можете использовать один и тот же класс в качестве шаблона для создания множества различных объектов.

Классы в PHP

В PHP класс может включать переменные, называемые свойствами для определения свойств объекта, и функции, называемые методами, для определения поведения объекта. Определение класса начинается с ключевого слова **class**, за которым следует имя класса. Фигурные скобки содержат определения свойств и методов, принадлежащих классу.

Классы в PHP

В PHP класс может включать переменные, называемые свойствами для определения свойств объекта, и функции, называемые методами, для определения поведения объекта. Определение класса начинается с ключевого слова **class**, за которым следует имя класса. Фигурные скобки содержат определения свойств и методов, принадлежащих классу.

Классы в PHP

```php
class Person {
  public $age; // свойство
  public function speak() { // метод
    echo "Hi!";
    }
}
```

Код выше определяет класс **Person**, который включает в себя свойство **age** и метод **speak()**.

Правильное имя класса начинается с буквы или символа подчеркивания, за которым следует любое количество букв, цифр или символов подчеркивания.

Обратите внимание на ключевое слово **public** перед методом **speak**, это обозначение видимости.
Ключевое слово **public** указывает на то, что метод может быть доступен из любого места в коде.
Есть и другие ключевые слова видимости, и мы узнаем о них в последующих уроках.

Задание № 1

Создайте класс **Student** с методом **sayHi**(), который должен выводить сообщение "**Hi!**" (без кавычек)

В качестве примера воспользуйтесь классом из предыдущего шага.

Объекты в PHP

Процесс создания объекта класса называется созданием экземпляра. Чтобы создать экземпляр класса, используйте ключевое слово **new,** как в примере ниже:

```
$bob = new Person();
```

В приведенном выше коде **$bob** является объектом класса **Person**.

Для доступа к свойствам и методам объекта используйте конструкцию со стрелкой (->), при этом заметьте, что при доступе к свойству не нужно указывать знак **$**, как в случае с обычной переменной:

Объекты в PHP

Для доступа к свойствам и методам объекта используйте конструкцию со стрелкой (->), при этом заметьте, что при доступе к свойству не нужно указывать знак **$**, как в случае с обычной переменной:

```
echo $bob->age;
```

В примере выше выводится значение свойства **age** для объекта **$bob**. Если вы хотите присвоить значение свойству, используйте оператор присваивания = как и для любой переменной.

Объекты в PHP

Давайте определим класс **Person**, создадим экземпляр класса, выполним присваивание и вызовем метод speak()

```php
class Person {
    public $age;
    public function speak() {
    echo "Hi!";
    }
}
$p1 = new Person(); //создание экземпляра класса
$p1->age = 23; // присваивание
echo $p1->age; // 23
$p1->speak(); // Hi!
```

Задание №2

Создайте объект "**dog**", который является экземпляром существующего класса "**Animal**".

$this

$this -** это псевдо-переменная, которая является ссылкой на вызывающий объект. При работе внутри метода используйте **$this точно так же, как и имя объекта вне класса.

```
class Dog {
    public $legs=4;
    public function display() {
    echo $this->legs;
    }
}
$d1 = new Dog();
$d1->display(); //4

$d2 = new Dog();
$d2->legs = 2;
$d2->display(); //2
```

Мы создали два объекта класса **Dog** и вызвали их методы display(). Так как метод display() использует **$this**, то значение **legs** ссылается на соответствующее значение свойства вызываемого объекта.

Как видите, каждый объект может иметь свои собственные значения для свойств класса.

Конструктор класса PHP

PHP предоставляет магический метод конструктора __construct(), который вызывается автоматически при создании нового объекта.

```
class Person {
    public function __construct() {
    echo "Object created";
    }
}
$p = new Person(); // Выведет: Object created
```

Конструктор класса PHP

Метод __construct() часто используется для любой инициализации, которая может понадобиться объекту перед его использованием. Параметры могут быть включены в __construct() для принятия значений при создании объекта.

```
class Person {
    public $name;
    public $age;
    public function __construct($name, $age) {
    $this->name = $name;
    $this->age = $age;
    }
}
$p = new Person("David", 42);
```

Конструктор класса PHP

```
class Person {
    public $name;
    public $age;
    public function __construct($name, $age) {
    $this->name = $name;
    $this->age = $age;
    }
}
$p = new Person("David", 42);
```

В приведенном коде конструктор использует аргументы в новом объекте для инициализации соответствующих свойств класса.

Нельзя писать несколько методов __construct() с разным количеством параметров. Различное поведение конструктора должно быть обработано с логикой в рамках одного метода __construct().

Задание № 3

Напишите класс **User** с использованием метода **__construct()**, который должен выводить слово "Конструктор" (без кавычек) при создании нового объекта этого класса.

Деструктор класса PHP

Подобно конструктору класса, существует магический метод-деструктор __destruct(), который автоматически вызывается при уничтожении объекта.

```
class Person {
    public function __destruct() {
    echo "Object destroyed";
    }
}
$p = new Person();
```

Деструктор класса PHP

Этот код создает новый объект класса **Person**. При завершении работы скрипта объект автоматически уничтожается, при этом вызывается деструктор и выводит сообщение "Object destroyed".
Чтобы явно вызвать деструктор, можно уничтожить объект с помощью функции unset() в выражении, подобном следующему: unset($p)

Деструкторы полезны для выполнения определенных задач при завершении жизненного цикла объекта. Например, освободить ресурсы, записать лог-файлы, закрыть соединение с базой данных и так далее.
PHP освобождает все ресурсы, когда скрипт завершает свое выполнение.

Задание 4 (тест)

Что будет выведено в результате выполнения следующего кода:

```
class TestMe {
    public function __construct() { echo "2"; }
    public function __destruct() { echo "1"; }
}
$test = new TestMe();
unset($test);
```

1) 2
2) 21
3) 12
4) 1

Наследование классов

Классы могут наследовать методы и свойства другого класса. Класс, который наследует методы и свойства, называется дочерним. Класс, от которого наследует дочерний класс, называется родительским классом.

Наследование осуществляется с помощью ключевого слова **extends**.

```
class Animal {
    public $name;
    public function hi() {
    echo "Hi from Animal";
    }
}
class Dog extends Animal {
}

$d = new Dog();
$d->hi(); // Выведет: Hi from Animal
```

Наследование классов

Здесь класс **Dog** наследуется от класса **Animal**. Как видите, все свойства и методы класса **Animal** доступны для объектов класса **Dog**.

Конструкторы родительских классов не вызываются неявно, если дочерний класс определяет конструктор. Однако, если дочерний класс не определяет конструктор, то он будет наследоваться от класса-родителя, если только он не объявлен как **private**.

Обратите внимание, что все наши свойства и методы имеют свойство видимости **public**.
Для добавления контроля над объектами, объявите методы и свойства, используя ключевое слово видимости. Это управляет тем, как и откуда можно получить доступ к свойствам и методам.
Посмотрите следующий урок, чтобы узнать больше о видимости.

Задание № 5

Напишите класс **Singer**, который бы наследовал уже созданный класс **Musician**

Видимость

Видимость контролирует, как и откуда можно получить доступ к свойствам и методам.
До сих пор мы использовали ключевое слово public, чтобы указать, что свойство/метод доступно откуда угодно.

Есть еще два ключевых слова для объявления видимости:
protected: делает элементы доступными только внутри самого класса, по наследованию и по родительским классам.
private: делает элементы доступными только в том классе, который их определяет.

Свойства класса всегда должны иметь тип видимости. Методы, объявленные без явного ключевого слова видимости, определяются как **public**.

Элементы с видимостью **protected** используются при наследовании.
Элементы с видимостью **private** используются только внутри класса.

Задание № 6 (тест)

Какое из перечисленных ключевых слов определяет тип видимости, разрешающий доступ только из дочерних классов?

1) Invisible
2) Private
3) Final
4) Protect

Интерфейсы (Interfaces)

Интерфейс задает список методов, которые должен реализовать класс. Однако сам интерфейс не содержит реализаций методов. Это важный аспект интерфейсов, так как он позволяет работать с методом по-разному в каждом классе, использующем интерфейс.

Ключевое слово **interface** определяет интерфейс.
Ключевое слово **implements** используется в классе для реализации интерфейса.

Интерфейсы (Interfaces)

Например, **AnimalInterface** определен с объявлением для функции **makeSound**(), но не реализуется до тех пор, пока не используется в классе:

```php
<?php
interface AnimalInterface {
    public function makeSound();
}

class Dog implements AnimalInterface {
    public function makeSound() {
    echo "Woof! <br />";
    }
}
class Cat implements AnimalInterface {
    public function makeSound() {
    echo "Meow! <br />";
    }
}

$myObj1 = new Dog();
$myObj1->makeSound(); // Выведет: Woof!

$myObj2 = new Cat();
$myObj2->makeSound(); // Выведет: Meow!
?>
```

Интерфейсы (Interfaces)

Класс может реализовать несколько интерфейсов. Несколько интерфейсов можно указать, разделив их запятыми.

```
class Demo implements AInterface, BInterface, CInterface {
  // Функции объявленые интерфейсами должны быть определены здесь
}
```

Интерфейс может быть унаследован другим интерфейсом с помощью ключевого слова **extends**.

Все методы, указанные в интерфейсе, требуют тип видимости **public**.

Задание № 7

У вас есть интерфейс **IMusician**. Реализуйте этот интерфейс в классе **Guitarist**. Функция в этом классе должна выводить "**playing a guitar**"

Абстрактные классы

Абстрактные классы могут быть наследованы, но не могут быть представлены в виде объектов.
Их преимущество заключается в том, что они могут содержать как методы с определениями, так и абстрактные методы, которые не определены до тех пор, пока они не будут наследованы.

Класс, наследуемый от абстрактного класса, должен реализовывать все абстрактные методы.

Абстрактные классы

Ключевое слово **abstract** используется для создания абстрактного класса или абстрактного метода.

```php
<?php
abstract class Fruit {
    private $color;

    abstract public function eat();

    public function setColor($c) {
    $this->color = $c;
    }
}

class Apple extends Fruit {
    public function eat() {
    echo "Omnomnom";
    }
}

$obj = new Apple();
$obj->eat(); // Выведет: Omnomnom
?>
```

Задание № 8

Что выведет следующий код:

```php
<?php
abstract class Calc {
    abstract public function calculate($param);
    protected function getConst() { return 4; }
    }
    class FixedCalc extends Calc {
    public function calculate($param) {
        return $this->getConst() + $param;
    }
    }
    $obj = new FixedCalc();
    echo $obj->calculate(38);
?>
```

Ключевое слово Static

Ключевое слово **static** определяет статические свойства и статические методы.
Доступ к статическим свойствам/методам класса может быть получен без создания объекта этого класса.

Доступ к статическому свойству или методу осуществляется с помощью оператора разрешения области видимости **::** между именем класса и именем свойства/метода.

```php
<?php
class myClass {
    static $myStaticProperty = 42;
}

echo myClass::$myStaticProperty; // Выведет: 42
?>
```

Ключевое слово Static

Ключевое слово **self** необходимо для доступа к статическому свойству из статического метода в определении класса.

```php
<?php
class myClass {
    static $myProperty = 42;
    static function myMethod() {
    echo self::$myProperty;
    }
}

myClass::myMethod(); // Выведет: 42
?>
```

Объекты класса не могут получить доступ к статическим свойствам в классе, но могут получить доступ к статическим методам.

Задание 3

Напишите класс **Singer**, в котором объявите статическое свойство "**name**" равное "**John**", и статический метод toSing(), из которого нужно получить доступ к свойству "**name**" и вернуть его (используя **return**).

Ключевое слово final

Ключевое слово **final** определяет методы, которые не могут быть переопределены в дочерних классах. Классы, которые определены как **final**, не могут быть наследованы.

Этот пример демонстрирует, что метод **final** не может быть переопределен в дочернем классе:

```php
<?php
class myClass {
    final function myFunction() {
    echo "Parent";
    }
}
// Выведет ошибку, потому что метод final не может быть переопределен в дочернем классе
class myClass2 extends myClass {
    function myFunction() {
    echo "Child";
    }
}
?>
```

Ключевое слово final

Следующий код демонстрирует, что класс **final** не может быть наследуемым:

```php
<?php
final class myFinalClass {
}

// Выведет ошибку, потому что класс final не может быть наследован
class myClass extends myFinalClass {
}
?>
```

*В отличие от классов и методов, свойства не могут быть отмечены ключевым словом **final**.*

Рефлексия

1) Что мы прошли сегодня на занятии?
2) Какие ваши впечатления от новой темы?
3) Какие основные моменты вы можете выделить?
4) Как вы поняли, что такое ООП?
5) Какие различия есть между классом и объектом?
6) Что такое $this?
7) Зачем нужно ключевое слово final?

Спасибо за внимание