# Лабораторна робота 6.

# Об'єктно-орієнтоване програмування на PHP

**Теоретичні відомості**

Стало стандартом, що вивчення ООП завжди починається з поняття «класу». Давайте і ми не будемо відходити від цих правил. Ідея об'єктно-орієнтованого програмування полягає в тому, що в ООП ми починаєм думати сутностями. А що таке сутність? - Це все, що нас оточує, тобто це звичайні об'єкти. А об'єктом у нас може бути все що завгодно (автомобіль, будинок, комп'ютер і т.д.). Що таке клас? - Клас описує ці об'єкти, тобто є їх формальною моделлю.

Наприклад, у нас є об'єкт «автомобіль». Перш ніж його зібрати, нам потрібне креслення. У цьому кресленні має прописуватися, те, що ми повинні знати про цей автомобіль - це те, що автомобіль має деякі властивості (колір, розмір, максимальна швидкість, рівень безпеки) і методи (автомобіль може рухатися вперед і назад, повертати кермо, стежити за втомою водія і т.д.).

В об'єктно-орієнтованому програмуванні для того, щоб описати таке креслення існує поняття «клас». «Клас» обов'язково повинен включати в себе опис таких понять, як властивості і методи даного об'єкту.

Отже, як же записується «клас»? Записується він у такий спосіб - спочатку йде ключове слово class, потім ім'я класу і фігурні дужки, в яких описуються властивості і методи:

<?php

class Car{

// Опис властивостей

// Опис методів

}

?>

Зверніть увагу, що імена класів чутливі до регістру.

Ґрунтуючись на цьому класі можна створювати об'єкти або в ООП вони ще називаються «екземпляри класу». Об'єкт класу створюється наступним чином - ставиться змінна, далі йде ключове new і ім'я класу з круглими дужками:

<? Php

// Опис об'єкта

class Car {

// Опис властивостей

// Опис методів

}

// Створення об'єкта

$car = new Car ();

?>

# Від одного класу (креслення) можна створювати кілька примірників класу або об'єктів (машин):

# <? Php

# // Опис об'єкта

# class Car {

# // Опис властивостей

# // Опис методів

# }

# // Створення об'єкта

# $car1 = new Car ();

# $car2 = new Car ();

# ?>

# Тепер давайте розберемо, що ж таке властивості і методи об'єкта. Почнемо з властивостей. Це звичайні змінні. Але не зовсім. Якщо змінна знаходиться десь в коді, то це звичайна змінна. А якщо змінна знаходиться в класі, то ця змінна - властивість.

# Наприклад, у автомобіля є рік випуску. Відповідно ініціалізуєм змінну $year. Ми можемо присвоїти їй значення по замовчуванню, наприклад 2011. Також у автомобіля є максимальна швидкість, ініціалізуєм змінну $speed. Ще у автомобіля є марка, ініціалізуєм змінну $model. Всі ці змінні - це властивості об'єкта, тобто нашого автомобіля.

# Однак в об'єктно-орієнтованому програмуванні перед такими змінними-властивостями прийнято ставити модифікатори доступу. Що таке модифікатори доступу? Це ключові слова, які позначають можливість доступу або можливість звернутися з методів об'єкта або з примірників класу до властивостей цього об'єкта. Ці ключові слова записуються як public, protected, private.

# Доступ до властивостей і методів класу, оголошених як public (загальнодоступний), дозволений звідусіль. Модифікатор protected (захищений) надає доступ наслідуваним і батьківським класам. Модифікатор private (закритий) обмежує область видимості так, що тільки клас, де оголошений сам елемент, має до нього доступ.

# Поки що будемо використовувати модифікатор public:

# <? Php

# // Опис об'єкта

# class Car {

# public $year = 2003;

# public $speed;

# public $model;

# }

# // Створення об'єкта

# $car1 = new Car ();

# $car2 = new Car ();

# ?>

# Тепер розберемо, що у нас тут відбувається. Ми з класу створили два об'єкти або два примірники класу, які мають однакові властивості, два з яких не визначені ($speed, $model) і одну властивість, у якої задане значення по замовчуванню ($year = 2003).

# Щоб звернутися до однієї з властивостей створеного об'єкта, необхідно написати ім'я змінної, якій присвоєно об'єкт, далі поставити ось таку стрілку -> (тире і знак більше) і ім'я властивості, але без знака долара:

# <? Php

# // Опис об'єкта

# class Car {

# public $year = 2003;

# public $speed;

# public $model;

# }

# // Створення об'єкта

# $car1 = new Car ();

# $car2 = new Car ();

# echo $car1-> year; // звернення до властивості об'єкта

# ?>

# Далі присвоюємо першому і другому автомобілю швидкість і модель. Так як у нас немає значення швидкості за замовчуванням і немає значення моделі, то можна зробити наступним чином: $car1-> speed = 210, $car1-> model = «bmw», $car2-> speed = 260, $car2 -> model = «lexus»:

# <? Php

# // Опис об'єкта

# class Car {

# public $year = 2003;

# public $speed;

# public $model;

# }

# // Створення об'єкта

# $car1 = new Car ();

# $car1-> speed = 210; // присвоєння значення властивості об'єкта

# $car1-> model = "bmw"; // присвоєння значення властивості об'єкта

# $car2 = new Car ();

# $car2-> speed = 260; // присвоєння значення властивості об'єкта

# $car2-> model = "lexus"; // присвоєння значення властивості об'єкта

# echo $car2-> year; // звернення до властивості об'єкта

# ?>

# Методи

Як було зазначено вище, всередині класу є дві речі: перше - це властивості, і друге - це методи. Тепер детальніше зупинимося на них. Що таке методи об'єкта? Якщо говорити зовсім простою мовою, то це поведінка об'єкта.

Згадайте про об'єкт «машина». Машина може рухатися вперед і назад, а також вправо або вліво, вона може розвивати певну швидкість. Все це є методами, тобто те, що може робити даний об'єкт.

Метод - це та ж сама функція! І описується вона точно так само, як звичайна функція.

Тільки функція, яка лежить десь в коді, це звичайна функція, а функція, описана в класі (або по-іншому сказати, що знаходиться всередині класу) - це метод. Запам'ятайте це!

У всіх методів, так само, як і у властивостей є точно такі ж модифікатори доступу (public, protected, private). Якщо не вказати жоден з модифікаторів, то за замовчуванням буде вважатися модифікатор public. Однак, краще завжди вказувати один з них.

Давайте на прикладі розгляне, як створюється метод. Продовжимо розгляд на прикладі класу, де ми описували властивості об'єкта «машина». Усередині цього класу ми опишемо найпростіший метод і називатися він у нас буде function takeSpeed​​().

Отримати доступ до цього методу можна точно так само, як і до властивості, наприклад:

$car1-> takeSpeed ​​():

<? Php

// Опис об'єкта

class Car {

// властивості

public $year = 2003;

public $speed;

public $model;

// метод

public function takeSpeed ​​() {

// Тут метод щось робить

echo "Швидкість автомобіля =";

}

}

// Створення об'єкта

$car1 = new Car ();

// Отримуємо доcтуп до методу

$car1-> takeSpeed ​​();

$car2 = new Car ();

?>

Якщо запустити цей код, то у нас викличеться наш метод і буде виведено: «Швидкість автомобіля =».

Ось тут починається найцікавіше. Ви звернули увагу, швидкість якого автомобіля ми хочемо отримати? Логічно припустити, що раз ми звертаємося з екземпляра класу car1, то і швидкість ми хочемо отримати автомобіля bmw. Однак, в методі, який у нас відповідає за отримання швидкості автомобіля, не вказано якого саме об'єкту швидкість нам потрібна. У цьому випадку нам необхідно звернутися до властивості speed класу Car. Але вона у нас одна, а автомобіля (об'єкта) два.

Так ось, нам потрібно вказати методу, властивість (швидкість) якого об'єкта ми маємо на увазі.

Таким покажчиком в ООП є ключове слово $this.

І записується це так: $this-> speed.

<? Php

// Опис об'єкта

class Car {

// властивості

public $year = 2003;

public $speed;

public $model;

// метод

public function takeSpeed ​​() {

// Вказуємо методу, чию швидкість ми хочемо отримати

echo "Швидкість автомобіля =". $this-> speed;

}

}

// Створення об'єкта

$car1 = new Car ();

$car1-> speed = 210; // присвоєння значення властивості об'єкта

// Отримуємо доcтуп до методу

$car1-> model = "bmw"; // присвоєння значення властивості об'єкта

$car1-> takeSpeed ​​();

$car2 = new Car ();

$car2-> speed = 260; // присвоєння значення властивості об'єкта

// Отримуємо доcтуп до методу

$car2-> model = "lexus"; // присвоєння значення властивості об'єкта

?>

Тепер у нас вже виведеться: Швидкість автомобіля = 210. Тобто ми отримали швидкість об'єкта $car1 (bmw).

Тут важливо запам'ятати - щоб звернутися з методу до властивості об'єкта, ми звертаємося через ключове слово $this.

Давайте розглянемо ще один важливий момент - припустимо у нас в класі Car крім методу takeSpeed ​​(), є метод paintSpeed ​​(). Ми зробимо так, що метод paintSpeed ​​() буде виводити швидкість автомобіля, а метод takeSpeed ​​() буде викликати метод paintSpeed ​​(). Це досить поширена ситуація, що показує - для того щоб звернутися з методу до іншого методу, також використовується ключове слово $this. Поглянемо на код:

<? Php

// Опис об'єкта

class Car {

// властивості

public $year = 2003;

public $speed;

public $model;

// метод

public function takeSpeed ​​() {

// Звертаємося з методу takeSpeed ​​() до методу paintSpeed ​​()

$this-> paintSpeed ​​();

}

// метод

function paintSpeed ​​() {

// Вказуємо методу, чию швидкість ми хочемо отримати. Йде звернення з методу до властивості

echo "Швидкість автомобіля =". $this-> speed;

}

}

// Створення об'єкта

$car1 = new Car ();

$car1-> speed = 210; // присвоєння значення властивості об'єкта

// Отримуємо доcтуп до методу

$car1-> model = "bmw"; // присвоєння значення властивості об'єкта

$car1-> takeSpeed ​​();

$car2 = new Car ();

$car2-> speed = 260; // присвоєння значення властивості об'єкта

// Отримуємо доcтуп до методу

$car2-> model = "lexus"; // присвоєння значення властивості об'єкта

$car2-> takeSpeed ​​();

?>

**Отже, звернення з методу до методу і звернення з методу до властивості здійснюється через ключове слово $this.**

# Дивіться, що тут відбувається. Ще нічого серйозного не зробили, а вже якось багато всього понаписувано. Щось тут не так і взагалі це йде в розріз з ідеологією ООП. Чи не можна це скоротити? Можна, можливо. І надалі теж будемо скорочувати кількість коду. У зв'язку з чим, ми може це зробити?

# На даному етапі вивчення об'єктно-орієнтованого програмування з'являється таке поняття, як «конструктор класу». Що ж це таке?

# Конструктор класу - це спеціальний метод, який автоматично викликається в момент створення об'єкту. У PHP конструктор класу має спеціальну назву - подвійне підкреслення construct (function \_\_construct).

# Давайте з вами перевіримо, як викликається конструктор для кожного створеного об'єкта. Запустіть цей код і Ви побачите, що для кожного об'єкта викликається конструктор:

# <? Php

# class Users {

# public $name;

# public $login;

# public $password;

# // Створення конструктора

# function \_\_construct () {

# echo "<p> Конструктор викликався автоматично!";

# }

# // Створюємо методод getInfo ()

# function getInfo () {

# echo "<p> Name:". $this-> name. "<br>";

# echo "Login:". $this-> login. "<br>";

# echo "Password:". $this-> password. "<br>";

# }

# }

# $user1 = new Users ();

# $user1-> name = "Vasya";

# $user1-> login = "vas";

# $user1-> password = 123;

# // Виводимо метод getInfo ()

# $user1-> getInfo ();

# $user2 = new Users ();

# $user2-> name = "Petya";

# $user2-> login = "pet";

# $user2-> password = 321;

# // Виводимо метод getInfo ()

# $user2-> getInfo ();

# $user3 = new Users ();

# $user3-> name = "Vova";

# $user3-> login = "vov";

# $user3-> password = 456;

# // Виводимо метод getInfo ()

# $user3-> getInfo ();

# ?>

# Коли ми створювали об'єкт: $user1 = new Users (); ми ставили круглі дужки біля об'єкту Users(). Логіка нам підказує, що раз є круглі дужки, то вони для чогось потрібні. Так ось, ці круглі дужки біля об'єкту - це дужки самого конструктора. Тобто, якщо ми передамо в круглих дужках об'єкта якийсь параметр, то він потрапить в конструктор. Давайте краще подивимося, як це виглядає в коді. Ми в конструкторі вкажемо параметр у вигляді змінної і будемо її виводити, а коли створюємо об'єкт, в ньому вкажемо числа (1,2,3). Запустіть цей код і подивіться, що вийде:

# <? Php

# class Users {

# public $name;

# public $login;

# public $password;

# // Створюємо конструктор з параметром

# function \_\_construct ($number) {

# echo "<p> Конструктор викликався автоматично $number!";

# }

# // Створюємо методод getInfo ()

# function getInfo () {

# echo "<p> Name:". $this-> name. "<br>";

# echo "Login:". $this-> login. "<br>";

# echo "Password:". $this-> password. "<br>";

# }

# }

# $user1 = new Users (1);

# $user1-> name = "Vasya";

# $user1-> login = "vas";

# $user1-> password = 123;

# // Виводимо метод getInfo ()

# $user1-> getInfo ();

# $user2 = new Users (2);

# $user2-> name = "Petya";

# $user2-> login = "pet";

# $user2-> password = 321;

# // Виводимо метод getInfo ()

# $user2-> getInfo ();

# $user3 = new Users (3);

# $user3-> name = "Vova";

# $user3-> login = "vov";

# $user3-> password = 456;

# // Виводимо метод getInfo ()

# $user3-> getInfo ();

# ?>

# Запам'ятайте - все, що ми передаємо в дужки об'єкта, потрапляє в конструктор.

 Поняття “конструктор” є у всіх мовах програмування, але в PHP він позначається саме, як нижня \_\_construct. Це метод автоматично викликається. Для чого використовується конструктор? Для ініціалізації чого-небудь. Наприклад, коли створюється новий об'єкт, ми хочемо щоб, щось відбувалося (щось в сесію записувалося, в файл що-небудь записувалося). Тобто, щось відбувалося автоматично. Це дуже схоже на автозагрузку операційної системи. Завантажилася операційна система і разом з нею сталися якісь дії, завантажилися програми і т.д. Тут те ж саме.

Однак, поряд з конструктором у нас є ще один метод, який використовується, коли закінчується код, тобто, коли код відпрацював і об'єкти видаляються. Звичайно ми можемо видалити об'єкт і раніше, ніж закінчиться код. Наприклад, функцією unset (). Але нам потрібно, щоб у разі вилучення об'єктів, неважливо примусово ми видаляємо їх чи вони видаляються після завершення коду, щось відбувалося, як і при створенні об'єктів.

Для цього у нас є метод, який називається **деструктор. Цей метод автоматично викликається при видаленні об'єкта.** У PHP він позначається, як подвійне нижнє підкреслення destruct (\_\_destruct):

<? Php

class Users {

public $name;

public $login;

public $password;

// Створення конструктора

function \_\_construct ($number) {

echo "<p> Конструктор зголосився автоматично $number!";

}

// Створюємо деструктор

function \_\_destruct () {

echo "<p> Об'єкт видалився!";

}

// Створюємо методод getInfo ()

function getInfo () {

echo "<p> Name:". $this-> name. "<br>";

echo "Login:". $this-> login. "<br>";

echo "Password:". $this-> password. "<br>";

}

}

$user1 = new Users (1);

$user1-> name = "Vasya";

$user1-> login = "vas";

$user1-> password = 123;

// Виводимо метод getInfo ()

$user1-> getInfo ();

$user2 = new Users (2);

$user2-> name = "Petya";

$user2-> login = "pet";

$user2-> password = 321;

// Виводимо метод getInfo ()

$user2-> getInfo ();

$user3 = new Users (3);

$user3-> name = "Vova";

$user3-> login = "vov";

$user3-> password = 456;

// Виводимо метод getInfo ()

$user3-> getInfo ();

?>

Запустивши цей код, ми бачимо, що при видаленні об'єктів, а вони видаляються у нас тут автоматично, викликається деструктор із записом - Об'єкт пішов. У нас було три об'єкти - відповідно виводяться три рази запис - «Об'єкт видалився!».

Зверніть увагу, що деструктор має круглі дужки, але передавати параметри в ці круглі дужки не можна! Ще один важливий момент - порядок видалення об'єктів ніколи не визначений. Тобто у нас є три об'єкти і їх треба видалити. Так ось, в якому порядку вони будуть видалятися невідомо. Тому є таке правило - з деструктора не звертатися до інших об'єктів, так як на момент виклику об'єкта, він може бути вже видалений.

**Копіювання і передача об'єктів за посиланням**

Тепер давайте поговоримо ось про що - копіювання і передача об'єктів за посиланням, яке так чи інакше завжди присутній в ООП. У чому ідея всього цього? Дивіться: $object2 = $object1. Як Ви думаєте, що це означає? Тут все залежить від того, якої версії PHP використовують — 4 чи >=5. Бо в 4 версії PHP це означало копіювання. Тобто об'єкт 2 - це копія об'єкта 1. А якщо ми хотіли передати об'єкт за посиланням, то ми писали ось таку річ — $object2 = & $object1. Що ж це означає - передати об'єкт за посиланням? Я думаю, що всі знають, що таке ярлик на робочому столі в Windows - це посилання на файл. Так ось тут те ж саме. Якщо більш зрозуміліше, то це два імені одного і того ж об'єкта. Все це стосується тільки 4 версії PHP.

А що ж у нас 5 версії? А тут у нас ось цей вислів $object2 = $object1 вже буде посиланням. Тоді виникає питання, а як же мені тоді скопіювати об'єкт? А копіюється він за допомогою чарівного слова clone: ​​$object2 = clone $object1.

Зверніть увагу, що при копіюванні об'єкта, конструктор не викликається. Але ж при копіюванні створюється новий об'єкт і у нас може виникнути необхідність, щось автоматично виконати.

У цьому випадку ми можемо описати спеціальний метод - подвійне нижнє підкреслення clone (\_\_clone), який буде автоматично викликатися при копіюванні об'єктів:

<? Php

class Users {

public $name;

public $login;

public $password;

// Створення конструктора

function \_\_construct ($name, $login, $password) {

$this-> name = $name;

$this-> login = $login;

$this-> password = $password;

}

// Створюємо метод clone

function \_\_clone () {

echo "<p> Об'єкт скопійований!";

}

// Створюємо метод getInfo ()

function getInfo () {

echo "<p> Name:". $this-> name. "<br>";

echo "Login:". $this-> login. "<br>";

echo "Password:". $this-> password. "<br>";

}

}

$user1 = new Users ( "Vasya", "vas", "123");

// Виводимо метод getInfo ()

$user1-> getInfo ();

$user2 = new Users ( "Petya", "pet", "321");

// Виводимо метод getInfo ()

$user2-> getInfo ();

$user3 = new Users ( "Vova", "vov", "456");

// Виводимо метод getInfo ()

$user3-> getInfo ();

// Об'єкт $user4 копія об'єкта $user3

$user4 = clone $user3;

?>

# Хід роботи

Завдання 1:

1. створити клас Student з властивостями name, suname, group;
2. створити три об'єкти класу Student;
3. задати довільні значення властивостям для кожного з об'єктів.

Завдання 2:

1. в класі Student потрібно описати метод getInfo ();
2. метод getInfo () повинен виводити значення властивостей об'єкта;
3. викликати метод getInfo () для кожного об'єкта.

Завдання 3:

* 1. в класі Student необхідно описати конструктор;
  2. конструктор повинен задавати початкові значення властивостей name, suname, group;
  3. створити крім існуючих трьох об’єктів класу Student, додатково ще три об'єкти з використання конструктора;

Завдання 4:

* 1. в класі Student описати метод \_\_clone ();
  2. метод \_\_clone повинен задавати початкові значення властивостей за замовчуванням при копіюванні об'єктів;

1. створити сьомий об'єкт, скопіювавши один з наявних об'єктів;