Objektinis PHP

1 žingsnis:

Pirmas dalykas, kurį turime padaryti, tai sukurti du PHP failus:

- index.php
- Libs/MyLib.php (folderyje Libs)

OOP yra viskas apie modulinio kodo kūrimą, todėl mūsų į objektus orientuotas PHP kodas bus įtrauktas į atskirus failus, kuriuos mes sujungsime į vieną PHP puslapį naudodamiesi php 'includes'.

Šiuo atveju visi mūsų OO PHP kodai bus PHP faile:

- MyLib.php

OOP sukasi aplink konstrukciją, pavadintą "klasė". Klasės yra brėžiniai / šablonai, naudojami objektams apibrėžti.

2 žingsnis:

Sukurkite paprastą PHP klasę (MyLib.php)

Vietoj to, kad turėtumėte daugybės funkcijų, kintamųjų ir kodo, padrikai išmėtytų po failą, norėdami suprojektuoti savo PHP skriptus arba kodų bibliotekas OOP būdu, turėsite apibrėžti / kurti savo klases.

Jūs nustatote savo klasę, pradedant raktiniu žodžiu "class", po kurio nurodomas vardas, kurį norite pateikti savo naują klasę.

```
class MyLib {


 }
```

Pastaba: klasę pridedate, naudodamiesi garbanotais skliaustais ({}) kaip ir su funkcijomis.

3 žingsnis:

Įtraukite duomenis į savo klasę

Klasės yra php objektų brėžiniai - daugiau apie tai vėliau. Vienas iš didžiausių funkcijų ir klasių skirtumų yra tai, kad klasėje yra tiek duomenys (kintamieji), tiek funkcijos, sudarančios paketą, vadinamą: "objektas".

Kuriant klasėje esantį kintamąjį jis vadinamas "property".

```
class MyLib {
     var $name;
 }
```

Pastaba: klasėje esantys duomenys / kintamieji (pvz., *Var $ name;*) vadinami "savybėmis".

4 žingsnis:

Pridėti savo klasėje funkcijas / metodus

Tuo pačiu būdu, kaip klasėje sukuriami kintamieji (jie vadinami: savybės) gauna kitokį pavadinimą, funkcijos taip pat pavadinamos kitaip. Funkcijos kai jos yra sukurtos klasėje yra vadinamos "metodais".

Klasės metodai naudojami manipuliuoti savo duomenimis / savybėmis.

```
class MyLib {
     var $name;
     function set_name($new_name) {
         $this->name = $new_name;
     }
     function get_name() {
         return $this->name;
     }
 }
```

Pastaba: nepamirškite, kad klasėje kintamieji vadinami "savybėmis", o funkcijos vadinamos "metodais".

5 žingsnis:

Getter ir setter funkcijos

Sukūrėme dvi įdomias funkcijas / metodus: get_name() ir set_name().

Šie metodai remiasi įprasta OOP konvencija, kurią matote daugybėje kalbų (įskaitant Java ir Ruby), kur kuriate metodus "nustatyti" ir "gauti" klasės savybėms.

Yra toks susitarimas, kad gavimo ir nustatymo pavadinimai turėtų atitikti savybių pavadinimus.

```
class MyLib {
     var $name;
     function set_name($new_name) {
          $this->name = $new_name;
     }
     function get_name() {
          return $this->name;
     }
 }
```

Pastaba: atkreipkite dėmesį, kad pavadinimai "getter" ir "setter" sutampa su susijusios savybės pavadinimu.

Tokiu būdu, kai kiti PHP programuotojai nori naudoti savo objektus, jie žinos, kad jei turite metodą / funkciją, vadinamą "set_name()", bus savybė / kintamasis, pavadintas "name".

6 žingsnis:

Kintamasis "$this"

Jūs tikriausiai pastebėjote šią kodo eilutę:

```
$this->name = $new_name;
```

Tai yra integruotas kintamasis (įterptas į visus objektus), kuris nurodo į esamą objektą. Arba, kitaip tariant, $this yra specialus savireguliavimo kintamasis. Jūs naudojate $this, kad galėtumėte pasiekti savybes ir kviesti kitus klasės metodus.

```
function get_name() {
   return $this->name;
}
```

Pastaba: tai gali būti šiek tiek painu kai kuriems iš jūsų ... būtent todėl, kad jūs pirmą kartą matote vieną iš tų kintamųjų, kurie yra įdėti OO funkcijose, kurie automatiškai ką nors daro.

Kol kas, tiesiog pagalvokite apie tai kaip į specialų OO PHP raktinį žodį.

7 žingsnis:

Naudokite savo klasę savo pagrindiniame PHP puslapyje. (index.php)

Jūs niekada negalite kurti savo PHP klasių tiesiai pagrindiniuose php failuose - tai prieštarautų pačiam objektinio programavimo principui.

Vietoj to visada geriausia yra kurti atskirus php failus, kuriuose yra tik jūsų klasės. Jas galėsite pasiekti įtraukdami į savo pagrindinius php failus naudodami PHP "include" arba "require".

```
include 'class_lib.php';
```

Pastaba: atkreipkite dėmesį, kad dar nieko nedarėme su mūsų klase. Mes tai padarysime toliau.

8 žingsnis:

Instantiate / sukurkite savo objektą

Klasės yra php objektų brėžiniai / šablonai. Klasės iš tikrųjų netampa objektais, kol nepadarysime kažko, vadinamo: instantiation.

Kada jūs instantiate klasę, jūs sukuriate klasės instance ... tokiu būdu sukuriate objektą.

Kitaip tariant, instancijacija yra procesas, sukuriantis objekto **instance**.

```
include 'class_lib.php';
```

```php
$stefan = new person();
```

Pastaba: kintamasis $stefan tampa nuoroda į mūsų naujai sukurtą objektą. $stefan yra nuoroda, nes per $stefan, mes valdysime objektą.

Jei dabar paleisite PHP kodą, nematysite jokio kodo, kuris būtų rodomas puslapyje. To priežastis yra ta, kad mes tik sukūrėme objektą, bet nieko su juo dar nedarėme.

9 žingsnis:

Raktinis žodis "new"

Norėdami sukurti objektą iš klasės, turite naudoti "new" raktinį žodį.

Sukūrę / kurdami klasę, galite pasirinktinai pridėti skliaustus į klasės pavadinimą, kaip tai padaryta toliau pateiktame pavyzdyje. Kad būtų aišku, žemiau esančiame kodo galite pamatyti, kaip galima sukurti keletą objektų iš tos pačios klasės.

Iš PHP variklio požiūriu kiekvienas objektas yra jo vienas subjektas. Ar tai prasminga?

```php
include 'class_lib.php';
$stefan = new person();
$jim = new person;
```

Pastaba: kurdami objektą, klasės pavadinimo nedėkite į kabutes.

10 žingsnis:

Nustatykite objektų savybes

Dabar, kai mes sukūrėme du atskirus "person" objektus, mes galime nustatyti jų savybes taikydami sukurtus metodus (kūrėjus).

Atminkite, kad nors abu mūsų objektai ($stefan ir $ jim) yra grindžiami ta pačia "person" klase, jie yra visiškai skirtingi objektai.

```php
include 'class_lib.php';
$stefan = new person();
$jim = new person;
$stefan->set_name("Stefan Mischook");
$jimmy->set_name("Jimmy Waddles");
```

11 žingsnis:

Prieiga prie objekto duomenų

Dabar mes panaudosime gavimo metodus, kad galėtume pasiekti mūsų objektuose laikomus duomenis. Tai yra tie patys duomenys, kuriuos įterpėme į mūsų objektus naudojant priskyrimo metodus.

Prieiga prie klasės metodų ir savybių naudojama rodyklės (->) operatoriaus.

```php
include 'class_lib.php';
$stefan = new person();
$jim = new person;
$stefan->set_name("Stefan Mischook");
$jimmy->set_name("Jimmy Waddles");
echo "Stefan vardas:" . $stefan->get_name();
echo "Jimmy vardas:" . $jimmy->get_name();
```

Pastaba: rodyklių operatorius -> nėra tas pats, kuris naudojamas su asociacijų masyvais: =>.

12 žingsnis:

Tiesiogiai pasiekiamos savybės

Jums nereikia naudoti metodų, norint pasiekti objektų savybes; galite tiesiogiai patekti į juos naudodami rodyklių operatorių (->) ir kintamojo pavadinimą.

Pavyzdžiui: savybė $name (objekte $stefan) turi tokią reikšmę:

```php
$name= $stefan->name;
echo "Stefan vardas:" . $name;
```

Nors tai įmanoma, visgi tai bloga praktika, nes tai gali sukelti problemų vėliau. Turėtumėte naudoti gavimo metodus.

13 žingsnis:

Konstruktoriai

Visi objektai gali turėti specialų metodą, vadinamą "konstruktoriumi". Konstruktoriai leidžia inicijuoti objekto savybes, kai objektą sukuriate..

Pastaba: jei sukursite __construct () funkciją (tai yra jūsų pasirinkimas), PHP automatiškai iškvies __construct () metodą / funkciją, kai kursite objektą iš savo klasės.

"Sukonstruoti" metodas prasideda dviem apatiniais brūkšniais (__) ir žodžiu "construct".

```php
class person {
     var $name;
     function __construct($people_name) {
         $this->name = $people_name;
     }
    function set_name($new_name) {
        $this->name = $new_name;
    }
    function get_name() {
        return $this->name;
    }
}
```

Prisiminkite, kad objektiniame programavime:

- Funkcijos = metodai

- Kintamieji = savybės

Kadangi tai yra OO PHP pamoka, toliau naudosim OO terminiją.

14 žingsnis:

Sukurkite objektą su konstruktoriumi

Dabar, kai sukūrėme konstruktoriaus metodą, mes galime suteikti vertę vardui $ name, kai mes kurdami savo asmeninius objektus.

Jūs galite perduoti duomenis konstruktoriui, pateikdami argumentų sąrašą po klasės pavadinimo.

Pavyzdžiui:

```
$stefan = new person("Stefan Mischook");
```

Tai sutaupo mums laiko, nes nereikia kviesti set_name() metodo, ir kartu sumažinana kodo kiekį. Konstruktoriai yra dažni ir dažnai naudojami PHP, Java ir kt.

```
$stefan = new person("Stefan Mischook");
echo "Stefan vardas:" . $stefan->get_name();
```

Tai tik mažas pavyzdys, kaip OO PHP įdiegti mechanizmai gali sutaupyti laiko ir sumažinti kodo kiekį, kurį reikia parašyti. Mažiau kodo reiškia mažiau klaidų.

15 žingsnis:

Ribojimas prieigai prie nuosavybės naudojant "prieigos modifikatorius"

Vienas iš pagrindinių OOP principų yra "inkapsuliavimas". Idėja yra tokia, kad jūs sukuriate švaresnį geresnį kodą, jei apribojate prieigą prie savo objektų duomenų savybių.

Jūs apribojote prieigą prie klasės savybių naudodami kažką, vadinamą "prieigos modifikatoriais". Yra 3 prieigos modifikatoriai:

1. public
2. private
3. protected

"Viešas" yra defaultinis modifikatorius.

```
class person {
      var $name;
      public $ugis ;
      protected $asmens_kodas;
      private $pin_number;
      function __construct($people_name) {
          $this->name = $people_name;
      }
     function set_name($new_name) {
         $this->name = $new_name;
     }
     function get_name() {
         return $this->name;
     }
}
```

Pastaba: kai deklaruojate savybę su "var" raktiniu žodžiu, ji laikoma "public". Bet geriau "var" iš vis nenaudoti.

16 žingsnis:

Prieigos prie savybių ribojimas: 2 dalis

Kai jūs deklaruojate nuosavybę kaip "privačią", prieiga prie savybės turi tik ta pati klasė.

Kai savybė yra paskelbtas "protected", tik ta pati klasė ir klasės, kilusi iš tos klasės, gali prieiti prie savybės - tai turi įtakos paveldėjimui ... daugiau apie tai vėliau.

Savybes, paskelbtas "viešosiomis", neturi prieigos apribojimų, o tai reiškia, kad visi gali prieiti prie jų.

Kad pradėtumėte suprasti šį (tikriausiai) rūko OOP aspektą, išbandykite šį kodą ir žiūrėkite, kaip PHP reaguoja. Patarimas: skaitykite kodo komentarus, jei norite gauti daugiau informacijos:

```
    include 'class_lib.php';
    $stefan = new person("Stefan Mischook");
    echo "Stefan's full name:" .  $stefan->name;
    /*
    Kadangi $pinn_number buvo paskelbtas privačiu, tai
kodo eilutė
    sukurs klaidą. Išbandykite!
    */
    echo "Pasakyk man privačius dalykus:" .
$stefan->pinn_number;
```

Pastaba: jei bandysite pasiekti privačią savybę / kintamąjį už klasės ribų, gausite tai:

```
" Fatalinė klaida : negalima pasiekti privačios savybės person::$pinn_number in
..."
```

17 žingsnis:

Prieigos prie metodų apribojimas

Kaip ir savybės, galite valdyti prieigą prie metodų naudodami vieną iš trijų

prieigos modifikatorių:

1. public
2. private
3. protected

Kodėl mes turime prieigą prie modifikatorių?

Trumpai tariant: objektinis programavimo esmė yra kontrolė, todėl yra tikslinga kontroliuoti, kaip žmonės naudojasi klasėmis.

Prieigos modifikatorių ir kitų OO konstruktų priežastis gali būti sudėtinga suprasti ... ypač dabar pradžioje.

Tada mes galime (apibendrinti) ir pasakyti, kad daugelis OOP konstrukcijų egzistuoja pagal idėja, kuri daugeliui programuotojų leidžia dirbti kartu su projektu.

```
class person {
      var $name;
      public $ugis ;
      protected $asmens_kodas;
      private $pin_number;
      function __construct($people_name) {
          $this->name = $people_name;
      }
     function set_name($new_name) {
         $this->name = $new_name;
     }
     function get_name() {
         return $this->name;
      }
      private function get_pinn_number() {
          return $this->pinn_number;
      }
  }
```

Pastabos: Kadangi metodas get_pinn_number() yra "privatus", vienintelė vieta, kuria galite naudoti šį metodą, yra klasės viduje. Jį galima naudoti taip pat paprastai, kaip naudojant bet kokį kitą metodą. Jei norite naudoti šį metodą tiesiai savo PHP kode, turėtumėte paskelbti jį "viešu".

18 žingsnis:

Paveldimumas - pakartotinis kodo naudojimas OOP būdu

Paveldimumas yra esminė OOP funkcija / konstrukcija, kurioje galite naudoti vieną klasę, kaip bazę / pagrindą kitai klasei ar daugybei kitų klasių.

Kodėl?

Tai leidžia efektyviai pakartotinai naudoti jūsų bazinėje klasėje esantį kodą.

Tarkime, jūs norėjote sukurti naują "darbuotojo" klasę ... nes galime pasakyti, kad "darbuotojas" yra "asmuo", jis dalijasi bendromis savybėmis ir metodais.

Kokia viso šito prasmė?

Tokioje situacijoje paveldėjimas gali padaryti jūsų kodą lengvesnį, nes jūs pakartotinai naudojate tą patį kodą dviem skirtingomis klasėmis. Tačiau skirtingai nuo "senosios mokyklos" PHP:

1. Turite tik vieną kartą sukurti kodą.
2. Faktinis kodas yra pakartotinai naudojamas, gali būti pakartotinai naudojamas daugumoje klasių,
3. bet jis įvedamas tik vienoje vietoje Konceptualiai tai yra tokia rūšis
4. kaip PHP ().

Pažvelkite į PHP pavyzdžio pavyzdį:

```
class employee extends person
{
  function __construct($employee_name) {
      $this->set_name($employee_name) ;
  }
}
```

19 žingsnis:

Kartotinis kodas su paveldėjimu: 2 dalis

Kadangi klasė "darbuotojas" yra pagrįsta "asmeniu", "darbuotojas" automatiškai turi visas viešąsias ir saugomas (protected) "asmens" savybes ir metodus.

Kodas:

```
class employee extends person
{
  function __construct($employee_name) {
      $this->set_name($employee_name) ;
  }
}
```

Atkreipkite dėmesį, kaip mes galime naudoti "employee_name", nors mes neaprašėme šio metodo "darbuotojo" klasėje. Taip yra todėl, kad mes jau sukūrėme set_name() klasėje "person".

Pastaba: toliau person klasė bus vadinama "tėvų" klase, nes klasė "darbuotojas" yra paveldėjus jos savybes ir metodus. Kai jūsų projektai tampa sudėtingesni, šios klasės hierarchija gali tapti labai svarbi.

20 žingsnis:

Kartotinis kodas su paveldėjimu: 3 dalis

Kaip matote žemiau esančiame kodo fragmente, mes galime naudoti "get_name" mūsų "darbuotojo" objektui, nors jo ir neaprašėme (bet paveldėjome)

Kodas:

```
include 'class_lib.php';
$stefan = new person("Stefan Mischook");
echo "Stefan's full name:" .  $stefan->name;
$james = new employee("Johnny Fingers");
echo "--->" . $james->get_name();
```

Tai yra klasikinis pavyzdys, kaip OOP gali sumažinti kodo eilučių skaičių (nereikia dvigubai rašyti tuos pačius metodus), tuo pat metu išlaikant kodą modulinį ir daug lengviau prižiūrimą.

21 žingsnis:

Pagrindiniai metodai

Kartais (kai naudojamas paveldėjimas) gali tekti keisti, kaip metodas veikia iš tėvinės klasės.

Pavyzdžiui, tarkime, kad "employee" klasėje, tarkim, "set_name ()" metodas turėjo daryti kažką kitko, negu "person" klasėje.

Galima ignoruoti "person" klasės versiją set_name(), deklaruojant tą patį metodą "employee".

Kodo fragmentas:

```
class person
{
    protected function set_name($new_name) {
        if ($new_name != "Jimmy Two Guns") {
            $this->name = strtoupper($new_name) ;
        }
    }
}
class employee extends person
{
    protected function set_name($new_name) {
        if ($new_name == "Stefan Sucks") {
            $this->name = $new_name;
        }
    }
}
```

Atkreipkite dėmesį, kaip set_name() skiriasi "darbuotojo" klasėje iš versijos, esančios tėvų klasėje: "person".

22 žingsnis:

Pagrindiniai metodai: 2 dalis

Kartais jums gali prireikti prieiti prie tėvinės klasės versijos metodo. Pavyzdyje mes "overclocked" metodą set_name() "darbuotojo" klasėje. Dabar naudosime kodą:

```
person::set_name($new_name);
```

Kodas:

```
class person
 {
     var $name;
     function __construct($persons_name) {
         $this->name = $persons_name;
      }
      public function get_name() {
         return $this->name;
      }

      protected function set_name($new_name) {
          if ($this->name != "Jimmy Two Guns") {
             $this->name = strtoupper($new_name);
          }
     }
 }
 class employee extends person
 {
     protected function set_name($new_name) {
     if ($new_name ==  "Stefan Sucks") {
         $this->name = $new_name;
     }
```

```
        else if ($new_name ==  "Johnny Fingers") {
            person::set_name($new_name);
        }
        }

        function __construct($employee_name)
        {
            $this->set_name($employee_name);
        }
    }
```

Galima kreiptis į dabartinės klasės tėvą - naudodojant raktinį žodį "parent".

Kodas:

```
    protected function set_name($new_name)
    {
        if ($new_name ==  "Stefan Sucks") {
            $this->name = $new_name;
        }
        else if ($new_name ==  "Johnny Fingers") {
            parent::set_name($new_name);
        }
    }
```

Išvada

Mes tik palietėme OO PHP pagrindus. Bet turėtumėte turėti pakankamai informacijos, kad jaustumėtės patogiai judėti į priekį.

Atminkite, kad geriausias būdas iš tikrųjų turėti šią medžiagą nusileidžia, yra faktiškai rašyti kodą.

Užduotis

<?php

/*

1. Sukurti klasę Company

2. Sukurti klases (vaikus) SoftwareCompany ir ConstructionCompany

3. Sukurti klases (anūkus) Programmer ir Builder

4. Klasė Company turi turėti savybes: name, employees, turnover

5. Company objekto kūrimo metu savybės turi būti:

   name - atsitiktinis stringas iš a-z raidžių, atsitiktinio ilgio nuo 5 iki 12, pirma raidė didžioji

   employees - atsitiktinis dydis nuo 3 iki 100

   turnover - atsitiktinis dydis nuo 10000 iki 10000000

6. Klasė SoftwareCompany turi turėti savybes programingLanguages

7. SoftwareCompany objekto kūrimo metu savybės turi būti:

   programingLanguages - 3 atsitiktiniai dydžiai iš aibės {PHP, Pascal, Go, C++, JAVA, Phyton}

6. Klasė ConstructionCompany turi turėti savybes buildingObjects

7. ConstructionCompany objekto kūrimo metu savybės turi būti:

   buildingObjects - 4 atsitiktiniai dydžiai iš aibės {Houses, Bridges, Offices, Roads, Gardens, Railroads, Canals, Aqueduct, Stadions}

8. Klasė Programmer ir Builder turi turėti savybes: skills, name

9. Programmer objekto kūrimo metu savybės turi būti:

    name - atsitiktinis stringas iš a-z raidžių, atsitiktinio ilgio nuo 5 iki 12, pirma raidė didžioji

    skills - 2 iš 3-jų programingLanguages (ne iš aibės) atsitiktinai parinkti dydžiai

10. Builder objekto kūrimo metu savybės turi būti:

    name - atsitiktinis stringas iš a-z raidžių, atsitiktinio ilgio nuo 5 iki 12, pirma raidė didžioji

    skills - 1 atsitiktinai parinktas dydis iš aibės {electrician, plumber, tractor driver, engineer}

11. Visos savybės turi būti inkapsuliuotos ir tiesiogiai nepasiekiamos

12. Sukurti metodus:

    printInfo() - išvestų tvarkingai suformatuotą visą informaciją (visas savybes)

    addSkill($skill) pridėtų nurodytą savybę. Programmer atveju, jeigu skill nėra viena iš programingLanguages, pridėtų ir tenai

    bankrupt() - ištrinti visas savybes iki tuščio stringo arba 0, paliekant tik savybę name, paleisti printInfo() metodą ir išvesti pranešimą:

    '{name} is bankrupt. {employees} employees are now unemployed.' (kur {name} ir {employees} yra Company klasės savybės)

13. Įvykdžius metodą bankrupt(), metodas addSkill() turi veikti kitaip: nevykdyti jokių prieš tai suprogramuotų veiksmų, o vietoj to išvesti pranešimą:

'Company {name} is bankrupt.' (kur {name} yra Company klasės savybės)

14. Įvykdyti kodą:

*/

$programmer = new Programmer();

$builder = new Builder();

```
$programmer->printInfo();

$programmer->addSkill('Rusty');

$programmer->addSkill('PHP');

$programmer->addSkill('Pascal');

$programmer->addSkill('Go');

$programmer->addSkill('JAVA');

$programmer->addSkill('Phyton');

$programmer->printInfo();

$programmer->bankrupt();

$programmer->addSkill('PHP');

$programmer->printInfo();


$builder->printInfo();

$builder->addSkill('Truck Driver');

$builder->printInfo();

$builder->bankrupt();

$builder->addSkill('Tank Driver');

$builder->printInfo();
```