

Olio-ohjelmointi PHP:llä

- PHP:n nykyiset olio-ominaisuudet ovat muiden nykyaikaisten oliokielten tasolla.
- Tässä esitetään esimerkkien avulla keskeisimmät PHP:n olio-malliin liittyvät asiat. Oletuksena on, että osaat jo aiemmin perusteet olio-ohjelmoinnista jollakin kielellä.

Luokan määrittely ja olion käyttäminen

Esimerkki 1701

Määritellään luokka `Person` ja muodostetaan siitä olio `$ari`. Ohjelma tulostaa "*Haloo, nimeni on Ari!*". PHP:ssä on normaalit näkyvyysmääreet *public*, *protected* ja *private*. Olion ominaisuuteen tai metodiin viitataan `->`-notaatiolla.

Konstruktori merkitään avainsanalla `__construct`

```
1  <?php
2  / / person-olio.php
3
4  class Person {
5      private $name;
6
7      public function __construct($name) {
8          $this->name = $name;
9      }
10     public function print_name() {
11         echo "Terve, nimeni on {$this->name}! <br>";
12     }
13 }
14
15 $ari = new Person("Ari");
16 $ari->print_name();
17 ?>
```

Esimerkki 1702

Usein luokat kirjoitetaan omaan tiedostoonsa nimeämällä ne tyyliin `Luokka.class.php`. Kirjoitetaan `Player`- ja `Team`-luokat pelaajien ja joukkueiden käsittelemistä varten. Huomaa: `protected`-määreitä tarvitaan jatkossa esitettävässä perintäesimerkissä.

Player.class.php

```
1  <?php
2  // Player.class.php
3
4  class Player {
5      protected $name;
6      protected $nro;
7
8      public function __construct($name, $nro) {
9          $this->name = $name;
10         $this->nro = $nro;
11     }
12
13     public function get_name() {
14         return $this->name;
15     }
16
17     public function get_nro() {
18         return $this->nro;
19     }
20
21 }
22
23 ?>
```

Team.class.php

```
1  <?php
2  // Team.class.php
3
4  class Team {
5      private $name;
6      private $players = array();
7
8      public function add($player) {
9          $this->players[] = $player;
10     }
11 }
```

```

11
12     public function playercount() {
13         return count($this->players);
14     }
15
16     public function playerlist() {
17         return $this->players;
18     }
19 }
20
21 ?>

```

teampayers.php

Luodaan erikseen luokkia **Player** ja **Team** käyttävä ohjelma. Funktiolla `__autoload` voidaan tietyllä tavalla nimetyt luokat määritellä etukäteen automaattisesti ladattaviksi vasta sitten kun niitä tarvitaan.

```

1  <?php
2  // teampayers.php
3
4  // Tarvittavien luokkien automaattinen lataaminen tarvittaessa
5  function __autoload($class_name) {
6      require_once $class_name . '.class.php';
7  }
8
9  // Luokat voitaisiin ladata myös tyyliin:
10 // require_once("Player.class.php");
11
12
13 $joukkue = new Team();
14 $joukkue->add(new Player("Ari", 1));
15 $joukkue->add(new Player("Pasi", 2));
16
17 echo "Pelaajia: " . $joukkue->playercount() . " kpl<br>";
18
19 // Tulostetaan pelaajat pelinumeroineen
20 foreach ($joukkue->playerlist() as $pelaaja) {
21     echo "Pelaaja: " . $pelaaja->get_name() .
22         "[" . $pelaaja->get_nro() . "]<br>";
23 }
24
25 ?>

```

Ohjelman tulostus on:

Pelaajia: 2 kpl

Pelaaja: Ari[1]

Pelaaja: Pasi[2]

Perintä

Peritään **Player**-luokan toiminnallisuus luotavalle **Golfplayer**-luokalle. **Golfplayer**-luokkaan lisätään pelaajan tasoitus eli HCP. Huomaa `protected`-määreet **Player**-luokassa, jotta ao. ominaisuudet voidaan periä. Lisäksi esitelty luokan erikoismetodi `__toString`, jota voidaan käyttää olion tulostamiseen merkkijonona viittaamatta tiettyyn metodiin.

Golfplayer.class.php

```
1  <?php
2  // Golfplayer.class.php
3
4  require_once("Player.class.php");
5
6  class Golfplayer extends Player{
7      private $hcp;
8
9      public function __construct($name, $nro, $hcp) {
10         parent::__construct($name, $nro);
11         $this->hcp = $hcp;
12     }
13
14     function __toString() {
15         return "<h4>$this->name [$this->nro]</h4>" .
16             "<p>HCP: $this->hcp </p>";
17     }
18 }
19 ?>
```

golfplayers.php

Huomaa tässä `__toString()` -metodin käyttäminen riveillä 12 ja 13.

```

1  <?php
2  // golfplayers.php
3
4  function __autoload($class_name) {
5      require_once $class_name . '.class.php';
6  }
7
8  $golplayer1 = new Golplayer("Ari", 1, 54);
9  $golplayer2 = new Golplayer("Pasi", 2, 5);
10
11 // Käyttää luokan erikoismetodia __toString:
12 echo $golplayer1;
13 echo $golplayer2;
14
15 ?>

```

Ohjelman tulostus on

Ari [1]

HCP: 54

Pasi [2]

HCP: 5

PHP:n olio-ominaisuuksia listattuna

Konstruktorit ja destruktorit

Määritellään seuraavasti.

```

1  class Jokuluokka {
2      function __construct() {
3          echo "Konstruktoria suoritetaan";
4      }
5      function __destruct() {
6          echo "Destruktoria suoritetaan";
7      }
8  }

```

Näkyvyysmääreet

- **public:** Käytettävissä kaikkialla
- **private:** Käytettävissä vain luokassa, jossa muuttuja on määritelty. Ei voi käyttää aliluokissa.
- **protected:** Käytettävissä siinä luokassa, jossa muuttuja on määritelty ja kaikissa aliluokissa

Luokkametodit ja luokkamuuttujat

Luokkametodi (eli staattinen metodi; static method, class method) tekee halutun toimenpiteen saamiensa parametrien perusteella ja palauttaa tuloksen. **Sen kutsua ei osoiteta millekään luokan oliolle!** Näin ollen luokkametodilla ei ole käytössä this-viittaustakaan.

Luokkamuuttuja (staattinen muuttuja; static variable, class variable) toimii luokkametodin tavoin. Luokkamuuttujalla ei voi olla erillisiä arvoja luokasta luotuja olioita kohden. Luokkamuuttujalla on vain yksi arvo.

```
1 class JokuLuokka {  
2     static function tervehdys() {  
3         echo "Terve vaan terve!";  
4     }  
5 }  
6  
7 JokuLuokka::tervehdys();
```

Abstraktit luokat ja metodit

- Abstrakteista luokista **ei voi** luoda sellaisenaan olioita.
- Ne kuvaavat vain aliluokkiensa yhteisiä muuttujia ja metodeja.
- Jos luokassa on yksikin abstrakti metodi, pitää myös asianomainen luokka määritellä abstraktiksi.

```
1 abstract class Astiat {  
2     abstract function annaTilavuus();  
3 }
```

Rajapinnat

- Rajapinta luettelee metodit, joille tulee löytyä toteutus asianomaisen rajapinnan toteuttavista luokista.
- Rajapinta ei sisällä ominaisuuksia eikä metodien toteutuksia.

- Erilaisille astioille tulee voida esimerkiksi laskea tilavuus, mutta tilavuuden laskeminen tapahtuu erimuotoisille astioille eri tavalla.
- PHP:ssa luokka voi periä vain yhden kantaluokan. Luokalta voidaan kuitenkin vaatia useiden rajapintojen toteutusta.

```
1 interface Astia {  
2     function annaTilavuus();  
3 }  
4  
5 class Tynnyri implements Astia {  
6     function annaTilavuus($sade, $korkeus) {  
7         return 3.14 * $sade * $sade * $korkeus;  
8     }  
}
```

Final-luokat ja metodit

Final-luokka on luokka, jota ei voi periä. Final-metodia ei voi määritellä uudelleen aliluokissa.

Esimerkki A

```
1 final class JokuFinalLuokka {  
2     //...  
3 }  
4  
5 // Tämä ei onnistu  
6 class Peritty extends JokuFinalLuokka {  
7     //...  
8 }
```

Esimerkki B

```
1 class JokuLuokka {  
2     final function annaTiedot() {  
3         echo "Nimeni on $this->nimi";  
4     }  
5 }  
6  
7  
8 class Peritty extends JokuLuokka {  
9     //sama nimi, kun Final-metodilla, ei onnistu:  
10    function annaTiedot() {
```

```
11 |         echo "Olen Tapani ja tapani on näyttää napani...";
12 |     }
13 | }
```

Vakioarvot

Luokka voi sisältää vakioarvoja, joihin voidaan viitata luokan nimellä.

```
1 | class JokuLuokka {
2 |     const VAKIO = "Vakiopaine";
3 | }
4 | echo JokuLuokka::VAKIO;
```

Ilmentymän toteaminen

instanceof-operaattorilla voidaan tarkistaa, onko joku olio tietyn luokan ilmentymä (instanssi)

```
1 | if ($olio instanceof Tynnyri) {
2 |     echo '$olio on Tynnyri';
3 | }
```

Olion kloonaaminen

Olio voidaan kloonata avainsanalla clone. Luokkaan voidaan määritellä myös `__clone()` -metodi, jota kutsutaan kloonauksen yhteydessä.

```
1 | class Lammas {
2 |     function __clone() {
3 |         print "Lampaan kloonaus käynnissä";
4 |     }
5 | }
6 |
7 |
8 | $dolly = new Lammas();
9 | dolly_copy = clone $dolly;
```


Tyypivihjeet

Tyypivihjeillä (type hints) voidaan esimerkiksi vaatia, että tietty metodi saa parametrinaan oikean tyyppisen luokan. Muussa tapauksessa kutsu aiheuttaa virheen.

```
1 | function odotaJokuLuokka(JokuLuokka $olio) {  
2 | }
```

© #AriRantala