Fakultet elektrotehnike, računarstva
i informacijskih tehnologija Osijek

Web programiranje

LV5: OOP PHP i MYSQL

ver 1.0 | Autor: Luka Omrčen

Contents

1 Uvod

Za mnoge programere objektno orijentirano programiranje predstavlja složen koncept, s puno kompliciranе sintakse i drugih prepreka. Objektno orijentirano programiranje stil je kodiranja koje omogućuje razvojnim inženjerima da grupiraju ponavljajući set radnji unutar definiranih klasa. Navedeni princip naziva se i "nemoj se ponavljati" (DRY).Jedna od glavnih predosti DRY principa je ta da ukoliko je potrebno promijeniti način izvođenja programa, umjesto mijenjanja većeg dijela programa potrebna je samo određena promjena za ažuriranje koda. Jedan od najvećih izazova programiranja predstavlja održavanje koda gdje se podaci i radnje deklariraju iznova i iznova, što znači da svaka promjena programa postaje nova dodatna frustracija koja vrlo lako može izmaknuti kontroli.

2 Include i Require

Ključna riječ **include** (ili require) uzima sadržaj koji postoji u određenoj datoteci i kopira je u datoteku koja ju poziva. Uključivanje datoteka vrlo je korisno kada je potrebno definirati isti PHP, HTML ili tekst na više različitih mjesta. U PHP-u postoje četiri ključne riječi u tu svrhu:

- include
- require
- include_once
- require_once

Ključne riječi **require** i **include** su identične, osim u načinu rukovanja greškama kada datoteka nije pronađena.

- **include** definira upozorenje i nastavlja izvršavanje skripte
- **require** stvara fatalnu grešku i zaustavlja izvršenje skripte

Ako je uključena skripta potrebna za sljedeće procese koji se definiraju, potrebno je koristiti **require**. Korištenjem naredbe **_once** provjerava se je li ta datoteka već uključena unutar trenutne skripte. Ako nije, datoteka će biti uključene. U suprotnom će se naredba preskočiti.

```
<html>
<body>
<?php require_once 'header.php' ?>

<p>
  Main content text
</p>

<?php include 'footer.php' ?>
<?php require_once 'footer.php'; ?> // ova naredba se ignorira posto je footer
    vec ucitan
</body>
</html>
```

Kod 1: Primjer korištenja include i require

3 Objektno orijentirani PHP

3.1 Klase

Unutar PHP-a za definiranje klase koristi se ključna riječ **class**. Postoje određena pravila prilikom deklariranja klase:

- naziv klase ne smije biti rezervirana ključna riječ,
- klasu treba započeti slovom ili doljnjom crtom,
- nakon prvog znaka klasa može imati slova, brojeve ili doljnu crtu.

Prije nego što definiramo svojstva klase, potrebno je označiti tip svojstva klase kako bi definirali dostupnost svojstva.

```
<?php
class House {
  public $primaryColor = 'black';
  public $secondaryColors = [
    'bathroom' => 'white',
    'bedroom' => 'light pink',
    'kitchen' => 'light blue'
  ];
  public $hasPool = false;
  public $extra;
}
```

Kod 2: Primjer definiranja klase

Iz definirane klase moguće je stvoriti više objekata. Objekti se nazivaju instancije klase. U primjeru koda prikazano je instanciranje objekta i dohvaćanje vrijednosti. Simbol → predstavlja operator pridruživanja, koji se koristi za pristup svojstvima i metodama unutar objekta.

```
<?php
$myHouse = new House();
echo $myHouse -> primaryColor;
$myHouse -> primaryColor = 'red';
echo $myHouse -> primaryColor;
```

Kod 3: Instanciranje objekta i dohvaćanje vrijednosti

3.2 Metode

U OOP jeziku, kao što je PHP, se metode nalaze unutar klasa. Deklaracija i ponašanje gotovo su slični standardnim funkcijama, osim što imaju posebnu primjenu unutar klase. Primjer koda 4 prikazuje definiranje metode i dohvaćanje putem instanciranog objekta. Kao i kod svojstva, prije metode potrebno je definirati tip metode kako bi definirali dostupnost navedene metode.

```
<?php
class Example {
```

```
  public function helloWorld($string) {
    echo $string;
  }
}

$example = new Example();
$example -> helloWorld('Hello World');
```

Kod 4: Definiranje metode i dohvaćanje unutar objekta

U primjeru koda 5 definirana je metoda koja mjenja vrijednosti svojstva. Unutar metode korištena je ključna riječ $this - pseudo varijabla (također rezervirana ključna riječ) koja je dostupna samo unutar metoda. Odnosi se na objekt trenutne metode.

```
<?php
class House {
  public $primaryColor = 'black';
  public function changeColor($color) {
    $this -> primaryColor = $color;
  }
}

$myHouse = new House();
echo $myHouse -> primaryColor;

$myHouse -> changeColor('white');
echo $myHouse -> primaryColor;
```

Kod 5: Definiranje metode i korištenje ključne riječi $this

3.3 Konstruktor i destruktor

U PHP-u je unutar klase moguće definirati konstruktore i destruktore.

- **Konstruktor**: poziva se kada se objekt stvori iz klase.
- **Destruktor**: zove se kada se objekt uništi, obično kada skripta završi s izvođenjem.

Konstruktor se definira ključnom riječi **__construct** i poziva se kada je objekt kreiran iz navedene klase. Pošto je konstruktor vrsta metode unutar klase, potrebno je dodjeliti tip podatka - preporučeno je da konstruktor uvijek bude public tipa.

```
<?php
class House {
  public $name;
  public $color;
  public function __construct($name, $color) {
    $this -> name = $name;
    $this -> color = $color;
  }
```

```
  public function showColor() {
    echo "The color of the {$this -> name} is {$this -> color}";
  }
}
$myHouse = new House("Zvonimir House", "gray");
$myHouse -> showColor();
```

Kod 6: Definiranje konstruktora

Za razliku od konstruktora, destruktor se zove kad je objekt uništen ili je skripta zaustavljena.

```
<?php
class House {
  public $name;
  public $color;
  public function __construct($name, $color) {
    $this -> name = $name;
    $this -> color = $color;
  }
  public function __destruct() {
    echo "Destructor called with this output: {$this -> name} is {$this -> color
    }";
  }
}
$myHouse = new House("Zvonimir House", "black");
```

Kod 7: Definiranje destruktora

3.4 Vidljivost podataka

Vidljivost određenog člana klase (svojstva, metode ili konstante) definira način pristupanja njegovoj vrijednosti unutar ili izvan klase. Postoje tri vrste vidljivosti:

- **Public** - pristupa se s bilo kojeg mjesta
- **Private** - moguće je pristupiti samo unutar klase
- **Protected** - mogući pristup iz klase u kojoj je deklarirana i klase koja nasljeđuje navedenu deklariranu klasu.

Privatna svojstva ili metode mogu se definirati dodavanjem ključne riječi **private** ispred svoje deklaracije. Članu privatne klase može se pristupiti samo iz okoline koja se nalazi unutar same klase. Kod 6 prikazuje dohvaćanje vrijednosti privatnog svojstva klase pomoću metode public tipa tzv. "getter" metode. Prilikom direktnog dohvaćanja svojstva PHP će vratiti pogrešku prilikom pozivanja naredbe.

```
<?php
class User {
  private $name = 'Zvonimir';
  public function getName() {
    echo $this -> name;
  }
```

```
  public function setName($name) {
    $this -> name = $name;
  }
}
$user = new User();
$user -> getName(); // vrijednost privatne klase dohvaceno preko "getter" metode
echo $user -> name; // error
$user -> setName("Mirko");
$user -> getName();
```

Kod 8: Dohvaćanje public i private tipa podatka

Zaštićeno svojstvo ili metoda mogu se definirati dodavanjem **protected** ključne riječi ispred deklaracije. Zaštićenim svojstvima i metodama se može pristupiti putem klase koja je definirala svojstvo i klase koja nasljeđuje deklariranu klasu.

```
<?php
class Example {
  protected $property = 'Protected value';
  protected function myMethod() {}
}
```

Kod 9: Protected tip podatka

3.5 Nasljeđivanje

U objektno-orijentiranom programiranju, kada klasa proizlazi iz druge klase, naziva se nasljeđivanjem. Derivirana klasa predstavlja dijete, dok klasa iz koje je dijete izvedeno je roditeljska klasa (ponekad se nazivaju podklasa ili super klasa). Drugim riječima, dječja klasa proširuje roditeljsku klasu. Podklasa nasljeđuje sva javna i zaštićena svojstva i metode od svoje super klase. Uz nasljeđivanje, podklasa može imati svoje definirane svojstva i metode. Podklase također mogu naslijediti druge podklase.

Primjer 10 prikazuje jednostavno nasljeđivanje između dvije klase. Nasljeđivanje se definira ključnom riječi **extends**.

```
<?php
class Person {
  public $name;
  public $age;
  public function __construct($name, $age) {
    $this -> name = $name;
    $this -> age = $age;
  }
  public function introduce() {
    echo "My name is {$this -> name}. My age is {$this -> age}";
  }
}
class Zvonimir extends Person {
  // __construct() metoda je nasljedena
```

```
  // introduce() metoda je nasljedena
  public function sayHello() {
    echo "Hello, World <br>";
  }
}
$zvonko = new Zvonimir('Zvonimir', 29);
$zvonko -> sayHello();
$zvonko -> introduce();
```

Kod 10: Dohvaćanje public i private tipa podatka

Definiranjem svojstva ili metode zaštićenim uvjetima pomoću **protect** moguće je ograničiti vidljivost između navedenih klasa.

```
<?php
class ParentClass {
  protected $protectedValue = 'Protected value';
  private $privateValue = 'Private value';
  protected function protectedMethod() {
    echo $this -> protectedValue;
  }
  private function privateMethod() {
    echo $this -> privateValue;
  }
}
class Child extends ParentClass {
  public function doSomething() {
    $this -> protectedMethod(); // Protected value
    $this -> privateMethod(); // error
  }
}
$child = new Child();
$child -> doSomething();
// echo $child -> protectedValue; // error
// echo $child -> privateValue; // error
// $child -> protectedMethod(); // error
// $child -> privateMethod(); // error
```

Kod 11: Ograničavanje vidljivosti između dvije klase

Naslijeđene metode mogu se "modificirati" tako što je potrebno unutar podklase definirati metodu s istim nazivom koja se nalazi unutar roditeljske klase.

```
<?php
class Person {
  public $name;
  public $age;
  public function __construct($name, $age) {
    $this -> name = $name;
    $this -> age = $age;
```

```
  }
  public function introduce() {
    echo "My name is {$this -> name}. My age is {$this -> age}";
  }
}
class Zvonimir extends Person {
  public $school;
  // konstruktor i metoda introduce su "modificirane"
  public function __construct($name, $age, $school) {
    $this -> name = $name;
    $this -> age = $age;
    $this -> school = $school;
  }
  public function introduce() {
    echo "My name is {$this -> name}. My age is {$this -> age}. My school is {
    $this -> school}";
  }
}
$zvonko = new Zvonimir('Zvonimir', 29, "School Legija");
$zvonko -> introduce();
```

Kod 12: Modificiranje metoda iz podklase

Iz podklase klase moguće je pozivati roditeljske metode pomoću ključne riječi **parent** i operator **::** (operator okoline). Operator okoline posebna je sintaksa u PHP-u koja se koristi u nekim slučajevima OOP-a.

```
<?php
class Person {
  public $name;
  public $age;
  public function __construct($name, $age) {
    $this -> name = $name;
    $this -> age = $age;
  }
  public function introduce() {
    echo "My name is {$this -> name}. My age is {$this -> age}";
  }
}
class Zvonimir extends Person {
  public $school;
  // konstruktor i metoda introduce su "modificirane"
  public function __construct($name, $age, $school) {
    // $this -> name and $this -> age definirani su iz roditeljskog konstruktora
    parent::__construct($name, $age);
    $this -> school = $school;
  }
  public function introduce() {
    echo "My name is {$this -> name}. My age is {$this -> age}. My school is {
    $this -> school}";
```

```
  }
}
$zvonko = new Zvonimir('Zvonimir', 29, "School Legija");
$zvonko -> introduce();
```

Kod 13: Pozivanje metode iz roditeljske klase pomoću parent i operatora okoline

Prilikom modificiranja roditeljske metode, nije moguće koristiti $this -> nazivMetode kako bi se pozvali na roditeljsku metodu. Navedena metoda se odnosi na metodu podređene klase. Kako bi pozvali metodu iz roditeljske klase potrebno je koristiti - **parent::nazivMetode**.

Ponekad postoji potreba za sprječavanjem nasljeđivanja određenih metoda ili klasa u objektno orijentiranoj hijerarhiji. U PHP-u za navedenu svrhu koristi se ključna riječ **final**.

```
<?php
final class Parent {
  // kod klase
}
class Child extends NonParent {} // error
////////////////////////////////////////
class Parent {
  final public function parentMethod() {}
}
class Child extends ParentClass {
  public function myMethod() {} // Error
}
```

Kod 14: Ograničavanje nasljeđivanja klase i određenih metoda

3.6 Apstraktne klase

Klase definirane kao apstraktne ne mogu biti instancirane, a bilo koja klasa koja u sebi sadrži barem jednu apstraktnu metodu mora biti definirana kao apstraktna. Metode definirane kao apstraktne definiraju značenje programeru da ne mogu implementirati navedenu metodu direktno. Prilikom nasljeđivanja apstraktne klase, podklasa mora definirati sve metode koje se nalaze u deklaraciji roditeljske klase. Navedene metode moraju se definirati s istom (ili manje ograničenom) vidljivošću.

Kako bi klasa ili metoda bila apstraktna potrebno je dodati ključnu riječ **abstract** u deklaraciji te ona postaje apstraktna. Apstraktne metode nemaju tijelo i zbog toga se ne koriste uglate zagrade {} (Kod 15).

```
<?php
abstract class Parent {
  abstract public function myMethod1();
  abstract protected function myMethod2($name, $age);
  abstract protected function myMethod3() : int;
}
```

Kod 15: Definiranje apstraktne klase

Podklasa koja nasljeđuje apstraktnu klasu treba "modificirati" sve apstraktne metode. Argumenti za navedene metode trebaju biti isti kao apstraktna metoda. Vidljivost djetetove metode trebala bi biti ista kao roditeljska ili manje ograničena (Kod 16).

- Apstraktna metoda Public ⇒ unutar podklase Public.
- Apstraktna metoda Protected ⇒ unutar podklase Protected ili Public.

```
<?php
abstract class Parent {
  abstract public function myMethod1();
  abstract protected function myMethod2($name, $age);
  abstract protected function myMethod3() : int;
}

class Child extends Parent  {
  public function myMethod1() {...}
  public function myMethod2($name, $age) {...}
  public protected myMethod3() : int {...}
}
```

Kod 16: Modificiranje abstraktne klase

Metode koje nisu apstraktne mogu se definirati u apstraktnoj klasi. Navedene metode funkcioniraju jednako kao i uobičajene metode koje se koriste prilikom nasljeđivanja. U primjeru koda 17 podklase će naslijediti konstruktor i svojstvo apstraktne klase jer je riječ o public metodi.

```
<?php
abstract class Person {
  public $name;
  public function __construct($name) {
    $this -> name = $name;
  }
  abstract public function hello() : string;
}

class Student extends Person {
  public function hello() : string {
    return "I'm " . $this -> name;
  }
}

class Teacher extends Person {
  public function hello() :string {
    return "Good morning";
  }
}

$student = new Student('Bobi');
```

```
echo $student -> hello();

$teacher = new Teacher('Rudi');
echo $teacher -> hello();
```

Kod 17: Modificiranje abstraktne klase

3.7 Sučelja

Sučelje omogućava razvojnim arhitektima definiranje metoda koje klasa mora sadržavati, bez uključivanja složenosti i detalja koje se provode unutar implementacije određene metode. Sučelja predstavljaju veću razinu apstrakcije i omogućava bolju sistematizaciju prilikom razvoja rješenja. Sučelje se definira slično kao i klasa, samo s ključnom riječi **interface**.

Sučelja moraju imati potpise navedenih metoda. Sve metode imaju samo deklaraciju i moraju biti public tipa (Kod 18). Pri uporabi sučelja vrijede sva pravila apstrakcije kao i kod apstraktnih klasa.

```
<?php
Interface Person {
  public function __construct($name);
  public function hello() : string;
}

class Student implements Person {
  public $name;
  public function __construct($name) {
    $this -> name = $name;
  }
  public function hello() : string {
    return "Hello from " . $this -> name;
  }
}
```

Kod 18: Definiranje sučelja i klase prema sučelju

Klasa može implementirati više sučelja odvojenih zarezima u deklaraciji (nakon ključne riječi za implementaciju).

```
<?php
Interface MyInterface1 {
  public function myMethod1();
}
Interface MyInterface2 {
  public function myMethod2();
}
class MyClass implements MyInterface1, MyInterface2 {
  public function myMethod1() {
    echo "Hello ";
  }
  public function myMethod2() {
```

```
    echo "World";
  }
}
$obj = new MyClass();
$obj -> myMethod1();
$obj -> myMethod2();
```

Kod 19: Definiranje više sučelja

Također je moguće implementirati jedno ili više sučelja unutar podklase. Potrebno je znati da je moguće proširiti samo jednu klasu istovremeno, za razliku od sučelja gdje je moguće definirati više sučelja (Kod 20).

```
<?php
Interface MyInterface {
  public function write();
}
class Parent {
  public $name;
  public function __construct($name) {
    $this -> name = $name;
  }
}
class Child extends Parent implements MyInterface {
  function write() {
    echo $this -> name;
  }
}
$child = new ChildClass('Mirko');
$child -> write();
```

Kod 20: Korištenje sučelja unutar nasljeđene klase

Sučelja također imaju mogućnost nasljeđivanja drugih sučelja.

```
<?php
Interface MyInterface1 {
  public function myMethod1();
}
Interface MyInterface2 extends MyInterface1 {
  public function myMethod2();
}
class MyClass1 implements MyInterface1 {
  public function myMethod1() {}
}
class MyClass2 implements MyInterface2 {
  public function myMethod1() {}
  public function myMethod2() {}
}
```

Kod 21: Nasljeđivanje sučelja

3.8 Osobine

Osobine su slične klasama, ali služe za kreiranje i grupiranje metoda na pouzdan način. Nije dopušteno samostalno označavati osobinu. Osobine se uvode kako bi se riješili problemi pojedinačnog nasljeđivanja klase. U slučaju svojstva, ono omogućava programeru da slobodno koristi skupove metoda u nekoliko neovisnih klasa koje žive u različitim hijerarhijama.

- Koristi se za definiranje metoda koje se mogu koristiti u više klasa.
- Smanjuje dupliciranje koda.
- Ne može se instancirati.
- Može imati metode i apstraktne metode.
- Metode mogu biti u bilo kojoj vidljivosti.

```
<?php
Trait Hello {
  public function hello() {
    echo "Hello";
  }
}
Trait World {
  public function world() {
    echo "World";
  }
}
class MyClass {
  use Hello, World;
}
$obj = new MyClass();
$obj -> hello();
$obj -> world();
```

Kod 22: Definiranje osobina

3.9 Konstante unutar klase

Ključna riječ **const** koristi se za definiranje konstanti unutar klase. Konstanta treba biti deklarirana unutar same definicije klase. Preporuka je imenovati konstante velikim slovima odvojeno doljnim crtama (_). Konstante nemaju modifikator vidljivosti.

Za dohvaćanje konstante unutar klase koristi se ključna riječ **self** i operator okoline za pristup konstantama. Kako bi dohvatili konstantu izvan klase potrebno je definirati naziv klase i naziv konstante operatorom okoline (Kod 23).

```
<?php
class Welcome {
```

```
  const HELLO = 'Hello World';
  public function hello() {
    echo self::HELLO;
  }
}
$welcome = new Welcome();
$welcome -> hello();

echo "<br>";
echo Welcome::HELLO;
```

Kod 23: Definiranje konstanti klase

Želimo li pristupiti bilo kojoj metodi i svojstvu klase bez objekta, tada te metode i svojstva moraju biti proglašena statičkim. Da bismo deklarirali statiku, moramo koristiti statičke ključne riječi prije naziva metode i entiteta. Statička ključna riječ također se naziva modifikator pristupa.

3.10 Statičke metode

Kako bi pristupili bilo kojoj metodi i svojstvu unutar klase bez instanciranja objekta, potrebno je da te metode i svojstva definiramo kao statičke. Ključna riječ **static** također se naziva modifikator pristupa. $this pseudo varijabla nije dostupna unutar statičkih metoda. Statičke metode i svojstva mogu se smatrati bijegom iz objektno orijentiranog programiranja, koje su ponekad potrebne.

Statičkim metodama može se pristupiti izvan klase pomoću naziva klase i operatora okoline (::). Statičke metode mogu se pozvati iz metoda drugih klasa na isti način.

```
<?php
class MyClass {
  public static function getName() {
    return 'My name iz Zvonko';
  }
}
echo MyClass::getName();
//////////////////////
class OtherClass { // Pozivanje staticke metode unutar metode druge klase
  public function getNameFrom() {
    MyClass::getName();
  }
}
```

Kod 24: Definiranje statičke metode unutar podklase

Ključna riječ **parent** unutar podklase klase odnosi se na roditeljsku klasu (Kod 25).

```php
<?php
class MyClass {
  protected static function getName() {
    return 'My name iz Zvonko';
  }
}
class MyExtendedClass extends MyClass {
  public $name;
  public function __construct() {
    $this -> name = parent::getName();
  }
}
$zvonko = new MyExtendedClass;
echo $zvonko -> name;
```

Kod 25: Definiranje statičke metode unutar podklase

3.11 Prostori

Kao i C ++, PHP prostori su jedan od načina enkapsuliranja stvari, tako da se ista deklarirana imena mogu ponovo koristiti bez sukoba.

- Omogućuje ponovno proglašavanje istih funkcija / klasa / sučelja / konstantnih metoda u zasebnom prostoru deklariranja imena.
- Prostor može sadržavati važeći PHP kod.
- Prostor utječe na klase, sučelja, funkcije i konstante.
- Prostori se deklariraju pomoću ključne riječi. **namespace**

Navedeni primjer u kodu 26 i 27 prikazuje prostor s definiranim metodama i klasom. Znak \ se koristi za spuštanje razine u definirane prostore.

```php
// Direktorij: ./src/Math/Constants.php
<?php
namespace Math;

class Constants {
  const PI = 3.14159;
}
```

Kod 26: Kreiranje prostora u direktoriju

U Circle.php koristimo **\Math\Constants::PI** kako bi referencirali na konstantu u datoteci Constants.php. PHP je uvijek relativan s trenutnim direktorijem prostora u kojem se nalazi. Kada prostor započne s kosom crtom (\), ime će biti dohvaćeno relativno u odnosu na globalni prostor imena.

- Ako bismo koristili Constants::PI u Circle.php, to bi se odnosilo na putanju **\Math\Geometry\Constants::PI** koja nije valjana jer Constants klasa se ne nalazi u Geometry direktoriju.
- Ako bismo u Circle.php koristili Math \Constants::PI (bez početne \), to bi se odnosilo na putanju **\Math\Geometry\Math\Constants::PI** koja nije valjana.

```
// Direktorij: ./src/Math/Geometry/Circle.php
<?php
namespace Math\Geometry;
class Circle {
  public $radius;
  public function __construct($radius) {
    $this -> radius = $radius;
  }
  public function getDiameter() {
    return $this -> radius * 2;
  }
  public function getArea() {
    // (pi)(r^2)
    return \Math\Constants::PI * $this -> radius ** 2;
  }
  public function getCircumference() {
    // 2(pi)(r)
    return 2 * \Math\Constants::PI * $this -> radius;
  }
}
```

Kod 27: Kreiranje i dohvaćanje prostora u poddirektoriju

Imena klasa relativna su u odnosu na trenutni prostor imena. Pomoću metode **include_one** pozivaju se dijelovi koda odvojeni prostorima, dok \NazivKlase predstavlja definiranu klasu.

```
// Direktorij: ./index.php
<?php
include_once 'src/Math/Constants.php';
include_once 'src/Math/Geometry/Circle.php';

$circle = new  Math\Geometry\Circle(10);
echo $circle -> getDiameter();     // 20
echo $circle -> getArea();       // 314.159
echo $circle -> getCircumference(); // 62.8318
```

Kod 28: Dohvaćanje i instanciranje koda odvojenog prostorima

Ključna riječ **use** se koristi za uvoz klase u trenutni okvir kako ne bi morali definirati \include metode i pomoću operatora \ definirati klase i metode iz željenog prostora.

```
// Direktorij: ./index.php
<?php
use Math\Geometry\Circle;

$circle = new Circle(10); // Circle klasa je dohvacena iz prostora Math\Geometry
     u navedeni okvir
```

Kod 29: Korištenje ključne riječi use

Moguće je uvesti prostore i klase zasebno (Kod 30).

```
// Direktorij: ./index.php
<?php
use Math\Geometry; // dohvacanje prostora
use Math\Geometry\Circle; // dohvacanje klase
```

Kod 30: Dohvaćanje prostora i klase prostora zasebno

Također, moguće je klasi iz drugog prostora promjeniti naziv unutar okvira u kojemu se nalazi - ključnom riječi **as** (Kod 31). Dva su najčešća pravila kada je potrebno promjeniti naziv uvežene klase:

- kada unutar okvira postoji klasu koja nosi isto ime kao i uvežena klasa,
- kada je potrebna prikladnija oznaka za uveženu klasu.

```
// Direktorij: ./index.php
<?php
use Math\Geometry\Circle as Circ;
$circle = new Circ(10);
```

Kod 31: Promjena imena uvežene klase iz danog prostora

4 PHP i MYSQL

Za navedeno poglavlje potrebno je pročitati materijale iz predavanja, AV-a i pročitati članak na poveznici [4]. Nakon pročitanog članka potrebno je proučiti (poželjno skinuti i isprobati) kod rješenja na poveznicama [1, 2, 3], s time da posebnu pažnju treba obratiti na projekt na poveznici [3].

References

[1] *OOP PHP i MYSQL github projekt 1*. URL: https://github.com/aamahi/Crud-OOP-PHP-OOP-mySql-.

[2] *OOP PHP i MYSQL github projekt 2*. URL: https://github.com/C-ALP/CRUD_PHP_OOP.

[3] *OOP PHP i MYSQL github projekt 3*. URL: https://github.com/MichelZuidema/PHP-OOP-CRUD.

[4] *PHP Object Oriented Programming (OOP) concept Tutorial with Example*. URL: https://www.guru99.com/object-oriented-programming.html.