Web-technológia II.

Objektum-orientált programozás PHP nyelven

Hatwágner F. Miklós

Széchenyi István Egyetem, Győr

2019. március 21.

Osztályok létrehozása

- Osztály létrehozása: class kulcsszóval, példányosítás: new
- Láthatósági kategóriák (visibility):

  private csak azonos osztály példányainak,
  protected azonos és leszármazott osztályok példányainak,
  public mindenhonnan engedélyezi az adattagok/metódusok elérését
- Kompatibilitás:
  - Adattag előtt var → public
  - Metódus előtt *nincs kulcsszó* → public
- Metódusok úgy használhatók, mint a függvények: type hinting, alapértelmezett értékek, stb.
- Adattagok elérése, metódusok hívása: $this-> (tehát nem =>, mint a tömböknél!)

Osztályok létrehozása

- Varázsmetódusok (magic methods): minden __ jellel kezdődő, mint pl:

  __construct konstruktor

  __destruct destruktor, erőforrások felszabadításához (pl. adatbázis kapcsolat lezárása)

  __toString objektumot karakterlánccá alakítva adja vissza

- Kompatibilitás: ha nincs __construct(), az osztály nevével egyező nevű konstruktort keres → metódus „véletlenül” konstruktorrá válhat!
- Konvenció: egy osztály – egy forrásfájl, nevek egyeznek → require_once()

Osztályok létrehozása

pelda23/Hallgato.php

```php
<?php
  declare(strict_types=1);

  class Hallgato {
    private $nev;
    private $neptun;

    public function __construct(string $nev = "[Gipsz Jakab]",
                                string $neptun = "[A1B2C3]") {
      $this->nev = $nev;
      $this->neptun = $neptun;
    }

    public function getNev() : string {
      return $this->nev;
    }

    public function getNeptun() : string {
      return $this->neptun;
    }
```

Osztályok létrehozása

pelda23/Hallgato.php

```
    public function setNev(string $nev) {
      $this->nev = $nev;
    }

    public function setNeptun(string $neptun) {
      $this->neptun = $neptun;
    }

    public function __toString() : string {
      return "Név: {$this->nev}, Neptun kód: {$this->neptun}";
    }
  }
```

Osztályok létrehozása

pelda23/main.php

```
<?php
  require_once("Hallgato.php");

  $istvan = new Hallgato("Trap Pista");
  $virag = new Hallgato("Cserepes Virág", "XYZ123");

  echo "<p>", $istvan->getNev(), " : ", $istvan->getNeptun(), "</p>\n";
  $istvan->setNeptun("QWERTZ");
  echo "<p>István új kódja: ", $istvan->getNeptun(), "</p>\n";
  echo "<p>", $virag, "</p>\n";

  $buta = new Hallgato(1.2, true); //implicit típuskonverziók
  echo "<pre>";
  var_dump($buta);
  echo "</pre>\n";
```

Varázsfüggvények

További varázsfüggvények → burkolóosztályok könnyen létrehozhatóak:

`__call` nem hívható metódus hívási kísérletekor

`__get` nem elérhető adattag olvasási kísérletekor

`__set` nem elérhető adattag írási kísérletekor

Metódushívásnál szükséges lehet a paramétereket tartalmazó tömb elemeinek felhasználása egy másik függvény hívásához:
`call_user_func_array()`

Varázsfüggvények

pelda24/File.php

```
<?php
  declare(strict_types=1);

  class File {
    private $fp = false;
    private static $functions = [
      "fgets", "gfetc", "feof"
    ];

    public function __construct(string $filename, string $mode) {
      $this->fp = fopen($filename, $mode);
    }

    public function __call(string $name, array $arguments) {
      if(in_array($name, self::$functions)) {
        return $name($this->fp);
      }
    }
```

Varázsfüggvények

pelda24/File.php

```
    public function __set(string $name, $value) {}

    public function __get(string $name) {
      return $this->$name;
    }

    public function __destruct() {
      if($this->fp != false) {
        fclose($this->fp);
      }
    }
  }
```

Varázsfüggvények

pelda24/main.php

```
<?php
  require_once("File.php");

  $f = new File("File.php", "r");
  echo "<pre>\n";
  while(!$f->feof()) {
    echo $f->fgets();
  }
  echo "</pre>\n";
  echo "<p>A fájlleíró értéke: ", $f->fp, "</p>\n";
```

Konstansok, statikus változók

Statikus adattagok

- deklaráció: static kulcsszóval
- $ jel a :: után!
- minősíteni kell

Minősítés:

- az osztály nevével, vagy
- self kulcsszóval (jobb, mert az osztály átnevezhető)

Konstansok

- deklaráció: const kulcsszóval
- nyilvános, statikusan tárolt, nincs $ jel a nevében!
- minősíteni kell

Konstansok, statikus változók

pelda25/Hallgato.php (részlet)

```
<?php
  declare(strict_types=1);

  class Hallgato {
    private $nev;
    private $neptun;
    const neptunHossz = 6;
    private static $lista = array();

    public function __construct(string $nev = "[Gipsz Jakab]",
                                string $neptun = "[A1B2C3]") {
      $this->nev = $nev;
      // Osztály nevével minősítve
      $this->neptun = substr($neptun, 0, Hallgato::neptunHossz);
      if(in_array($this->neptun, Hallgato::$lista)) {
        die("Nem létezhet két azonos kódú hallgató.");
      } else {
        Hallgato::$lista[] = $this->neptun;
      }
    }
```

Konstansok, statikus változók

pelda25/Hallgato.php (részlet)

```
public function setNeptun(string $neptun) {
  // self kulcsszó
  $idx = array_search($this->neptun, self::$lista);
  unset(self::$lista[$idx]);
  $regi = $this->neptun;
  $this->neptun = substr($neptun, 0, self::neptunHossz);
  if(in_array($this->neptun, self::$lista)) {
    $this->neptun = self::$lista[] = $regi;
    die("Nem létezhet két azonos kódú hallgató.");
  } else {
    self::$lista[] = $this->neptun;
  } }
```

pelda25/main.php (részlet)

```
$virag = new Hallgato("Cserepes Virág", "1234567890");
echo "<p>Virág Neptun kódja: ", $virag->getNeptun(), "</p>\n";
$virag->setNeptun("ABCDEFGHIJKLMNO");
echo "<p>Virág Neptun kódja: ", $virag->getNeptun(), "</p>\n";

// $pista = new Hallgato('Trap Pista', 'ABCDEF');
$pista = new Hallgato('Trap Pista', 'ABCDE2');
$pista->setNeptun('ABCDEF');
```

Névterek

- azonosítók (osztályok) csoportosítása
- több egyforma azonosító azonos névtérben nem fordulhat elő
- névtér szakaszok elválasztása: \ jellel
- szakaszok többé-kevésbé követik a könyvtárszerkezetet
- névtér deklarálás: namespace kulcsszóval
- névtér osztályának példányosítása:
  - osztály neve a névtérrel minősítve
  - use kulcsszó után a névtérrel minősített osztály megnevezése
- Ha eltérő névterekben azonos nevű osztályok vannak, és mindegyiket „use-oljuk?” → álnevek:

```
use FelsoOkt\Egyetem\SZE\Hallgato as SZEHallgato;
new SZEHallgato();
```

Automatikus osztálybetöltés

- mindig pontosan azokat az osztályokat tölti be, amikre szükség van, nem felejtjük el, ...
- varázsmetódus: __autoload(), paraméter a névtérrel minősített osztálynév
- probléma: egyszerre csak egy ilyen metódus létezhet → megoldás: készítsünk saját betöltőt, regisztráljuk egy központi listába, a rendszer pedig addig hívja ezeket, míg valamelyik nem lesz sikeres: spl_autoload_register()
- a betöltőknek kell gondoskodni a névtér → elérési út konverzióról!

Névterek és osztálybetöltők

pelda26/Hallgato.php (részlet)

```
<?php
  declare(strict_types=1);
  namespace Egyetem;
  class Hallgato {
    private $nev;
```

pelda26/main.php

```
<?php
  use Egyetem\Hallgato;
  /*
  function __autoload($osztaly) {
    require_once(substr($osztaly, strrpos($osztaly, '\\')+1).".php");
  }
  */
  spl_autoload_register(function ($osztaly) {
    require_once(substr($osztaly, strrpos($osztaly, '\\')+1).".php");
  });
  // $virag = new Egyetem\Hallgato("Cserepes Virág", "1234567890");
  $virag = new Hallgato("Cserepes Virág", "1234567890");
  echo "<p>", $virag, "</p>\n";
```

Származtatás, absztrakt osztályok

Legfontosabb szintaktikai szabályok:

- ős jelölése a leszármazott osztály deklarálásakor: extends
- csak egyszeres öröklődés támogatott
- hivatkozás öröklött metódusra: parent kulcsszóval
- absztrakt osztályok és metódusok létrehozhatóak az abstract kulcsszóval

Származtatás, absztrakt osztályok

pelda27/HTMLgen/Valasztok.php

```php
<?php
  declare(strict_types = 1);

  namespace HTMLgen;

  abstract class Valasztok {
    protected $name;
    protected $valueLabel;
    protected $checked;

    public function __construct(string $name, array $valueLabel) {
      $this->name = $name;
      $this->valueLabel = $valueLabel;
    }

    public function addValueLabel(string $value, string $label) {
      $this->valueLabel[$value] = $label;
    }

    public abstract function printHTML();
  }
```

pelda27/HTMLgen/RadioGomb.php

```
<?php
  declare(strict_types = 1);
  namespace HTMLgen;

  class RadioGomb extends Valasztok {
    public function __construct(string $name, array $valueLabel,
                                $checked=false) {
      parent::__construct($name, $valueLabel);
      $this->checked = $checked;
    }

    public function printHTML() {
      foreach($this->valueLabel as $value => $label) {
        echo "<div><label><input type=\"radio\"
              name=\"{$this->name}\" value=\"$value\"";
        if($value == $this->checked) echo " checked";
        echo ">$label</label></div>\n";
      }
    }
  }
```

Származtatás, absztrakt osztályok

pelda27/HTMLgen/JelolоNegyzet.php

```
<?php
  declare(strict_types = 1);
  namespace HTMLgen;

  class JelolоNegyzet extends Valasztok {
    public function __construct(string $name, array $valueLabel,
                                array $checked) {
      parent::__construct($name, $valueLabel);
      $this->checked = array_unique($checked);
    }

    public function printHTML() {
      foreach($this->valueLabel as $value => $label) {
        echo "<div><label><input type=\"checkbox\"
              name=\"{$this->name}[]\" value=\"$value\"";
        if(in_array($value, $this->checked)) echo " checked";
        echo ">$label</label></div>\n";
      }
    }
  }
```

Származtatás, absztrakt osztályok

pelda27/main.php

```
<?php
  use HTMLgen\RadioGomb;
  use HTMLgen\JelolоNegyzet;

  spl_autoload_register(function ($osztaly) {
    require_once(str_replace('\\', '/', $osztaly).".php");
  });

  $nem = new RadioGomb('nem', [
    'ffi' => 'férfi',
    'no'  => 'nő'
  ], 'no');

  $hobby = new JelolоNegyzet('hobby', [
    'olv' => 'olvasás',
    'moz' => 'mozi és filmek',
    'gas' => 'gasztronómia, főzés',
    'spo' => 'sport'
  ], [
    'moz', 'spo'
  ]);
```

pelda27/main.php

```
$hobby->addValueLabel('tur', 'túrázás');

if(!empty($_POST)) {
  echo "<pre>\n";
  var_dump($_POST);
  echo "</pre>\n";
}

echo "<form method=\"post\"",
  " action=\"{$_SERVER['PHP_SELF']}\">\n";
$nem->printHTML();
$hobby->printHTML();
echo "<div><input type=\"submit\"",
  " value=\"Küldés\"></div>\n</form>\n";
```

Interfészek

- származtatási kapcsolatban nem lévő, de azonos szolgáltatást (is) nyújtó osztályok számára; „követelménylista”
- kulcsszó: interface
- az interfész típusként használható → type hinting

pelda28/Vilag/iID.php

```
<?php
  declare(strict_types = 1);
  namespace Vilag;

  interface iID {
    public function getID() : string;
    public function hasID(string $id) : bool;
  }
```

Interfészek

pelda28/Vilag/Ember.php

```php
<?php
  declare(strict_types = 1);
  namespace Vilag;

  class Ember implements iID {
    private $szigSzam = array();

    public function __construct(string $szigSzam) {
      $this->szigSzam[] = $szigSzam;
    }
    public function addSzigSzam(string $szigSzam) {
      $this->szigSzam[] = $szigSzam;
    }
    public function getID() : string {
      array_push($this->szigSzam, ($akt = array_pop($this->szigSzam)));
      return $akt;
    }
    public function hasID(string $id) : bool {
      return in_array($id, $this->szigSzam);
    }
  }
```

pelda28/Vilag/Merevlemez.php

```php
<?php
  declare(strict_types = 1);
  namespace Vilag;

  class Merevlemez implements iID {
    private $serialNo;

    public function __construct(string $serialNo) {
      $this->serialNo = $serialNo;
    }

    public function getID() : string {
      return $this->serialNo;
    }

    public function hasID(string $id) : bool {
      return $id === $this->serialNo;
    }
  }
```

pelda28/Vilag/Tarolo.php

```php
<?php
  declare(strict_types = 1);
  namespace Vilag;

  class Tarolo {
    private $dolgok = [];

    public function addDolog(iID $uj) {
      $this->dolgok[] = $uj;
    }

    public function __toString() : string {
      $str = "<ul>";
      foreach($this->dolgok as $dolog) {
        $str .= "<li>".$dolog->getID()."</li>";
      }
      return $str."</ul>";
    }
  }
```

pelda28/main.php

```
<?php
  use Vilag\Ember;
  use Vilag\Merevlemez;
  use Vilag\Tarolo;

  spl_autoload_register(function ($osztaly) {
    require_once(str_replace('\\', '/', $osztaly).".php");
  });

  $giziElso = '123456AB';
  $gizi = new Ember($giziElso);
  $gizi->addSzigszam('654321BA');
  echo "<p>Gizi személyi igazolvány száma: ", $gizi->getID(), "</p>\n";
  echo "<p>Gizi használta már a $giziElso számú személyi igazolványt? ",
    $gizi->hasID($giziElso)?"Igen":"Nem", "</p>\n";

  $seagateSN = 'xyz123asd';
  $seagate = new Merevlemez($seagateSN);
  echo "<p>Merevlemez gyártási száma: ", $seagate->getID(), "</p>\n";
  echo "<p>Merevlemez rendelkezik a $seagateSN sorozatszámmal? ",
    $seagate->hasID($seagateSN)?"Igen":"Nem", "</p>\n";
```

Interfészek

pelda28/main.php

```
$kamra = new Tarolo();
$kamra->addDolog($gizi);
$kamra->addDolog($seagate);
echo "<p>Kamrában tárolt dolgok azonosítói: ", $kamra, "</p>\n";
```

Trait

Trait $\approx$ jellemvonás, jellemző

- nincs többszörös öröklődés $\rightarrow$ trait
- egymástól teljesen független osztályok azonosan implementált funkcionalitással (copy-paste helyett)
- egy trait több osztályba is beépülhet (hivatkozás use kulcsszóval)
- egy osztály több trait-et is használhat (vesszővel elválasztott lista)
- deklarálása: trait kulcsszóval
- trait tartalmazhat:
  - adattagot,
  - (absztrakt/statikus) metódust (akár statikus változóval),
  - felhasználhat (use) más trait-eket

Trait

Ütközésfeloldás

- azonos metódus több forrásból → precedencia
  1. aktuális osztály metódusa
  2. trait metódusa
  3. öröklött metódus
- több beépített trait azonos nevű metódussal
  - Csak az egyiket használjuk:
    `use Trait1, Trait2 {Trait1::metodus insteadof Trait2;}`
  - Álnevet használunk:
    `use Trait1, Trait2 {Trait1::metodus as t1metodus;}`

Láthatóság változtatása a hivatkozó osztályban

- `use Trait { metodus as protected; }`
- `use Trait { metodus as private sajatMetodus; }`

Forrás: StackOverflow

pelda29.php

```
<?php
  interface Logger
  {
    public function log($message, $level);
  }

  class DemoLogger implements Logger
  {
    public function log($message, $level)
    {
      echo "Logged message: $message with level $level", PHP_EOL;
    }
  }
```

Trait

pelda29.php

```
trait Loggable // implements Logger
{
  protected $logger;
  public function setLogger(Logger $logger)
  {
    $this->logger = $logger;
  }
  public function log($message, $level)
  {
    $this->logger->log($message, $level);
  }
}

class Foo implements Logger
{
  use Loggable;
}

$foo = new Foo;
$foo->setLogger(new DemoLogger);
$foo->log('It works', 1);
```

Hibák és kivételek

Osztályhierarchia:

Throwable → Error (belső PHP hibák, kezelhetőek mint a kivételek, ld. még set_exception_handler())

→ Exception

Az Exception legfontosabb metódusai:

__construct()	Konstruktor (üzenet, kód, előzmény)
getMessage()	üzenet lekérdezés
getCode()	hibakód lekérdezés
getPrevious()	előzmény lekérdezés
getFile()	a fájl, ahol a kivétel keletkezett
getLine()	a sor, ahol a kivétel keletkezett

Hibák és kivételek

Kulcsszavak:

try	az itt keletkezett hibák/kivételek kezelhetőek
catch(*Osztály* $változó)	adott típusú kivétel példány elfogása
finally	mindenképpen lefutó utasítások (pl. erőforrás felszabadítás)

Catch ágak sorrendje:

- legspeciálisabbak előre (*Osztály* minden leszármazottját is elfogja)
- az Error és az Exception testvérek, külön catch blokkok kellenek

Kivétel kiváltása: throw, a metóduson nem kell külön jelezni, hogy milyen típusú kivételt válthat ki (mint pl. Java-ban)
Előre definiált kivételek, SPL kivételek

Hibák és kivételek

pelda30/Hallgato.php (részlet)

```
<?php
  declare(strict_types=1);
  namespace Egyetem;

  class Hallgato {
    private $nev;
    private $neptun;
    const neptunHossz = 6;
    private static $lista = array();

    public function __construct(string $nev = "[Gipsz Jakab]",
                                string $neptun = "[A1B2C3]") {
      $this->nev = $nev;
      if(strlen($neptun) != Hallgato::neptunHossz) {
        throw new NeptunException("A neptun kód hossza nem megfelelő.");
      }
      if(in_array($neptun, Hallgato::$lista)) {
        throw new NeptunException('Ilyen neptun kód már létezik.'); }
      $this->neptun = $neptun;
      Hallgato::$lista[] = $neptun;
    }
```

Hibák és kivételek

pelda30/Hallgato.php (részlet)

```
    public function setNeptun(string $neptun) {
      if(strlen($neptun) != Hallgato::neptunHossz) {
        throw new NeptunException("A neptun kód hossza nem megfelelő.");
      }
      $idx = array_search($this->neptun, self::$lista);
      unset(self::$lista[$idx]);
      if(in_array($neptun, self::$lista)) {
        self::$lista[] = $this->neptun;
        throw new NeptunException('Ilyen neptun kód már létezik.');
      }
      $this->neptun = $neptun;
      self::$lista[] = $neptun;
    }
```

pelda30/NeptunException.php

```
<?php
  namespace Egyetem;
  class NeptunException extends \Exception {}
```

Hibák és kivételek

pelda30/main.php

```
<?php
  use Egyetem\Hallgato;
  use Egyetem\NeptunException;
  spl_autoload_register(function ($osztaly) {
    require_once(substr($osztaly, strrpos($osztaly, '\\')+1).".php");
  });
  try {
    // $virag = new Hallgato("Cserepes Virág", "1234567890");
    $virag = new Hallgato("Cserepes Virág", "123456");
    // $toni = new Hallgato("Trab Antal", "123456");
    $toni = new Hallgato("Trab Antal", "ABCDEF");
    // $toni->setNeptun('qwertzuiop');
    $toni->setNeptun('ABCDEF');
    // $toni->setNeptun('123456');
    $toni->nincs();
  } catch(NeptunException $ne) {
    echo 'NeptunException: ',$ne->getMessage();
  } catch(\Error $e) {
    echo 'Error: ',$e->getMessage();
  } finally {
    echo "<p>Ez a sor mindig látszik.</p>\n"; }
```

Programtervezési minták

„Receptek” tipikus feladatok megoldásához (PHP implementációk)
Gyár (factory) minta:

- ha valamiért el akarjuk rejteni a példányosítási folyamatot a felhasználó elől (függés verzióktól, platformtól, stb.)
- statikus metódus végzi a példányosítást, majd ad vissza valamilyen objektumot

pelda31/gyar.php

```
<?php
  declare(strict_types=1);

  interface iUser {
    public static function getUser() : string;
  }

  class UserOldPHP implements iUser {
    public static function getUser() : string {
      return isset($_GET['user']) ? $_GET['user'] : 'senki';
    }
  }
```

pelda31/gyar.php

```
class UserPHP7 implements iUser {
  public static function getUser() : string {
    return $_GET['user'] ?? 'senki';
  }
}

class UserFactory {
  public static function getInstance() : iUser {
    if(phpversion() < '7.0.0') {
      return new UserOldPHP();
    } else {
      return new UserPHP7();
    }
  }
}

$felh = UserFactory::getInstance();
echo "<p>Felhasználónév: ", $felh->getUser(), "</p>\n";
```

Egyke (singleton) minta:

- Antiminta?!
- Célja: egy osztályból pontosan egy példány létezzen, és mindenhol azt használják (pl. megosztott erőforrás kezelése)
- Minta: egyedi azonosító előállítása: garantálni kell az egyediséget, ezt egy közös objektum fogja megtenni

pelda31/egyke.php

```
<?php
  declare(strict_types=1);

  class IDgenerator {
    private $szamlalo;
    private static $peldany = NULL;

    public static function getInstance() : IDgenerator {
      if(self::$peldany === NULL) {
        self::$peldany = new IDgenerator();
      }
      return self::$peldany;
    }
```

pelda31/egyke.php

```
    private function __construct() {
      $this->szamlalo = -1;
    }

    public function getID() : string {
      return $this->id = 'ID'.++$this->szamlalo;
    }
  }

  // $id0 = new IDgenerator();
  $id1 = IDgenerator::getInstance();
  echo "<p>", $id1->getID(), "</p>\n";
  echo "<p>", $id1->getID(), "</p>\n";
  echo "<p>", $id1->getID(), "</p>\n";
  $id2 = IDgenerator::getInstance();
  echo "<p>", $id2->getID(), "</p>\n";
  echo "<p>", $id1->getID(), "</p>\n";
```

Szerializáció

Érték átalakítása karakterlánccá, vagy fordítva

- Erőforrások és néhány beépített objektum nem szerializálható. Függvények:
  - serialize() paraméterként adott érték átalakítása karakterlánccá
  - unserialize() paraméterként adott karakterlánc átalakítása változóvá
- utóbbi példányosítást és kódfuttatást vonhat maga után → felhasználó ne férjen hozzá szerializált adathoz, vagy használjunk JSON-t!
- példányosítás helyén az osztálynak ismertnek kell lennie
- a függvények implicit módon varázsfüggvényeket hívnak:
  - __sleep() „takarítás”
  - __wakeup() korábban megszakadt tevékenységek folytatása, pl. adatbázis kapcsolat ismételt megnyitása

Szerializáció

pelda32/main.php

```
<?php
  use Egyetem\Hallgato;
  spl_autoload_register(function ($osztaly) {
    require_once(substr($osztaly, strrpos($osztaly, '\\')+1).".php");
  });
  define('FAJL', '/home/feltoltes/www/hallgato.dat');

  $hg = NULL;
  if(file_exists(FAJL)) {
    echo "<p>Fájl létezik, betöltés...</p>\n";
    $hg = unserialize(file_get_contents(FAJL));
  } else {
    echo "<p>Fájl nem létezik, létrehozás...</p>\n";
    $hg = new Hallgato('Bekő Tóni', 'QWERTZ');
    file_put_contents(FAJL, serialize($hg));
  }
  echo "<p>Hallgató adatai: ", $hg, "</p>\n";
```

Feladatok

Készítsen könyvtár alkalmazást! A *Könyvtár Könyv* példányokat tárol. Minden könyvről nyilván kell tartani a következőket:

- cím
- szerző
- ISBN
- kulcsszavak listája

Lássa el az osztályokat lekérdező és beállító metódusokkal, konstruktorral és `__toString()` metódussal! Az adatmódosításnál és a konstruktornál végezzen minimális input ellenőrzést és kivételkezelést! (Pl. ISBN 13 karakter hosszú.) Készítsen olyan metódust a *Könyv* osztályban, ami megmondja, hogy a könyv hány kulcsszóval rendelkezik a megadottak közül!
A *Könyvtár* tárolja a Könyveket, lehetővé teszi a cím, szerző, ISBN vagy kulcsszó/szavak alapján történő keresést (listázás az egyező kulcsszavak száma szerint csökkenő sorrendben), könyvek felvitelét. A könyvek adatait hosszú távon is meg kell őrizni!

Felhasznált irodalom

Antonio Lopez
Learning PHP7
Pact Publishing Ltd., Birmingham, UK, 2016.

Hivatalos PHP referencia