WEBTERVEZÉS – GYAKORLATI JEGYZET

10. gyakorlat

PHP – Sütik, fájlfeltöltés,
objektumorientált programozás

Készítették:

Cservenák Bence

Farkas Anikó

Savanya Sándor

FIGYELEM!

A jegyzet folyamatosan készül, így előfordulhatnak benne apróbb hibák, hiányosságok, elírások. Ha valaki esetleg ilyet találna, kérem, írjon a `Cservenak.Bence@stud.u-szeged.hu` címre, hogy mihamarabb javíthassuk.

1. Sütik (Cookie-k)

- A böngészősütiket (cookie-kat) a menetkövetés megvalósítására használjuk
- Kis mennyiségű adatot tárolnak, kliensoldalon (böngészőben)
  - pl. eltárolhatjuk a kiválasztott nyelvet, témát stb., így később ezt nem kell ismét megadni
  - **NE** használjuk felhasználói adatok (pl. felhasználónév, jelszó) tárolására
    - mivel a sütik kliensoldalon tárolódnak, ezért bármikor szabadon szerkeszthetők
- A HTTP fejlécben továbbítjuk, ezért a tartalom elküldése előtt be kell állítani
- PHP-ban a `setcookie()` függvénnyel tudunk sütit létrehozni – paraméterei rendre:
  - süti neve (kötelező)
  - süti értéke (opcionális)
  - lejárati idő (opcionális)
    - általában az aktuális időponthoz (`time()` függvény) hozzáadott másodpercek számával adjuk meg
  - elérési útvonal a szerveren belül (opcionális)
  - domain név (opcionális)
  - továbbítás csak a HTTPS biztonságos protokollon keresztül (opcionális)
  - elérés csak HTTP-n keresztül (opcionális)
- PHP-ban a sütik a `$_COOKIE` szuperglobális tömbben tárolódnak

```php
<?php
   // "theme" nevű süti, "dark" értékkel, élettartama 1 nap
   setcookie("theme", "dark", time()+86400);
?>

<!DOCTYPE html>
<html>
   <head>
   </head>
   <body>
      Téma:
      <?php
         if ($_COOKIE["theme"] == "dark")
            echo " sötét";
         else
            echo " világos";
      ?>
   </body>
</html>
```

Példa: Süti használata – Eredménye: "Téma: sötét"

2. Fájlfeltöltés

- Ahhoz, hogy fájlokat tölthessünk fel a szerverre, a HTML-ben az alábbiaknak kell teljesülnie:
  - a `<form>` tag rendelkezik a `method="POST"` attribútummal
  - a `<form>` tag rendelkezik az `enctype="multipart/form-data"` attribútummal
  - az űrlapon található egy `<input type="file"/>` űrlapmező a fájlfeltöltéshez
- Rejtett mezők segítségével korlátozhatjuk kliensoldalon a fájl maximális méretét (bájtokban)
  - az `<input type="file"/>` mező **elé** írjuk:
    `<input type="hidden" name="MAX_FILE_SIZE" value="30000"/>`
  - böngészőfüggő, általában nem működik
  - a fájl méretét szerveroldalon is ellenőriznünk kell
- Az `<input type="file"/>` `accept` attribútumával megkötést tehetünk a fájl kiterjesztésére
  - a kiterjesztést szerveroldalon is ellenőriznünk kell

```
<body>
   <form action="index.php" method="POST" enctype="multipart/form-data">
      <input type="hidden" name="MAX_FILE_SIZE" value="100000000"/>
      <input type="file" name="image" accept="image/*"/>  <!-- csak képek -->
      <input type="submit" name="upload-btn" value="Feltöltés"/> <br/><br/>
   </form>
</body>
```

- PHP-ban a feltöltött fájlok adatait a `$_FILES` szuperglobális tömb tartalmazza
  - `name`: a feltöltött fájl neve
  - `tmp_name`: a fájl ezzel az ideiglenes névvel töltődik fel
  - `size`: a feltöltött fájl mérete (bájtokban)
  - `type`: a feltöltött fájl típusa
  - `error`: hibakód (0, ha nem történt hiba a feltöltés során)
- A fájl kiterjesztését szerveroldalon is lekérdezhetjük
  - az **`explode`**`($delim, $s)` függvény feldarabolja az `$s` stringet `$delim` karakterek mentén
    - `$darabok = explode('.', $_FILES["my_file"]["name"]);`
    - `$kiterjesztes = strtolower(end($darabok));`
  - használhatjuk a **`pathinfo`**`($path, PATHINFO_EXTENSION)` függvényt is
    - `$path` a feltöltött fájl neve (`$_FILES["my_file"]["name"]`)
- A **`move_uploaded_file`**`($t_name, $dest)` függvénnyel a `$t_name` ideiglenes névvel (`tmp_name`) rendelkező fájlt áthelyezzük az ideiglenes mappából a `$dest` helyre

Példa: A fenti kódpéldában szereplő űrlaphoz tartozó PHP kódpélda

```php
<?php
   if (isset($_POST["upload-btn"])) {
      $engedelyezett_kiterjesztesek = ["jpg", "jpeg", "png"];

      // fájl kiterjesztésének lekérdezése
      $kiterjesztes = pathinfo($_FILES["image"]["name"], PATHINFO_EXTENSION);

      if (in_array($kiterjesztes, $engedelyezett_kiterjesztesek)) {
         // ha a fájl kiterjesztése szerepel az engedélyezett kiterjesztések között…

         if ($_FILES["image"]["error"] === 0) {       // sikeres feltöltés…
            if ($_FILES["image"]["size"] < 100000000) {  // fájlméret ellenőrzés
               // útvonal (a fájl az images mappába kerül az eredeti fájlnévvel)
               $dest = "images/" . $_FILES["image"]["name"];

               // fájl áthelyezése a megadott helyre
               move_uploaded_file($_FILES["image"]["tmp_name"], $dest);
               echo "Sikeres fájlfeltöltés <br/>";
            } else {
               echo "A fájlméret túl nagy! <br/>";
            }
         } else {
            echo "Hiba történt a fájl feltöltése közben! <br/>";
         }
      } else {
         echo "Nem megfelelő kiterjesztés! <br/>";
      }
   }
?>
```

3. Objektumorientált programozás

A PHP támogatja az objektumorientált paradigmát.

A. Osztályok, objektumok

- **Osztályok**
  - osztályokat a **`class`** kulcsszóval hozhatunk létre
  - adattagokat (tagváltozók) és metódusokat (tagfüggvények) tartalmaz
  - az adattagok és metódusok elérése a -> operátorral történik
  - láthatósági módosítószók (access specifiers)
    - **`private`**: csak az adott osztályból látható
    - **`protected`**: csak az adott osztályból és annak gyermekosztályaiból látható
    - **`public`**: mindenhonnan látható (alapértelmezett)

- **konstruktor**: speciális metódus, ami az osztály példányosításakor fut le
  - szintaxisa PHP-ban: `__construct($param1, $param2, …) { … }`
  - az adattagok inicializálására használjuk
  - `$this` segítségével az aktuális objektumra hivatkozhatunk
- **destruktor**: speciális metódus, ami az osztálypéldány megszűnésekor fut le
  - szintaxisa PHP-ban: `__destruct() { … }`
- **getter**: adattagok értékének lekérdezésére szolgáló metódus
- **setter**: adattagok értékének beállítására szolgáló metódus

```php
<?php
   class Ember {

      // adattagok

      private $nev;
      private $eletkor;
      protected $foglalkozas;

      // konstruktor + destruktor

      public function __construct($nev, $eletkor, $foglalkozas) {
         $this->nev = $nev;               // adattagok inicializálásra
         $this->eletkor = $eletkor;
         $this->foglalkozas = $foglalkozas;

         echo "--- " . $this->nev . " létrehozva. --- <br/>";
      }

      public function __destruct() {
         echo "--- Viszlát " . $this->nev . "! --- <br/>";
      }

      // getterek + setterek

      public function getEletkor() {      // get_nev, get_foglalkozas ugyanígy…
         return $this->eletkor;
      }

      public function setEletkor($eletkor) { // set_nev, set_foglalkozas ugyanígy…
         if ($eletkor > 0)
            $this->eletkor = $eletkor;
      }

      // metódusok

      public function bemutatkozik() {
         echo "Szia! " . $this->nev . " vagyok, " . $this->eletkor . " éves. <br/>";
      }

      // ...
   }
?>
```

- **Objektumok**
  - az általunk írt osztályok példányai
  - a példányosítás (objektumok létrehozása) a **new** kulcsszóval történik
    - ekkor meghívjuk az osztály konstruktorát
    - pl. `$objektumNeve = new OsztalyNeve( /* konstruktor paraméterek */ );`

```php
<?php
   class Ember {

      // ide jön a fenti kód…

   }

   // példányosítsuk az Ember osztályunkat

   $ember1 = new Ember("Jónás", 21, "maffiózó");
   $ember2 = new Ember("Márk", 14, "YouTuber");

   echo $ember1->getEletkor() . "<br/>";        // getter teszt
   $ember2->setEletkor(25);                     // setter teszt

   $ember1->bemutatkozik();   // a bemutatkozik() metódus tesztelése
   $ember2->bemutatkozik();

   // a kód végén lefut a destruktor…
?>
```

```
--- Jónás létrehozva. ---
--- Márk létrehozva. ---
21
Szia! Jónás vagyok, 21 éves.
Szia! Márk vagyok, 25 éves.
--- Viszlát Márk! ---
--- Viszlát Jónás! ---
```

  - objektumok összehasonlítása:
    - == operátor igazat ad vissza, ha a két objektum adattagjainak neve és értéke rendre megegyezik
    - === operátor igazat ad vissza, ha a két objektum ugyanaz az objektum

```php
<?php
   class Ember {
      // ...
   }

   $ember1 = new Ember("Károly", 20, "gyártulajdonos");
   $ember2 = new Ember("Károly", 20, "gyártulajdonos");
   $ember3 = $ember1;

   var_dump($ember1 == $ember2);    // bool(true)
   var_dump($ember1 === $ember2);   // bool(false)
   var_dump($ember1 === $ember3);   // bool(true)
?>
```

B. Objektumok másolása

- Vegyük a fenti kódpéldában az `$ember3 = $ember1` értékadást
  - az `$ember3` egy referencia, ami ugyanarra az objektumra mutat, mint az `$ember1`
  - ekkor nem jön létre új objektum
- Ha viszont azt szeretnénk, hogy új objektum jöjjön létre, használjuk a **klónozást**
  - klónozó metódus szintaxisa PHP-ban: `__clone() { … }`
  - objektum klónozása: `$obj2 = clone $obj1;`
  - módosíthatjuk a létrejövő objektum adattagjait

```php
<?php
   class Ember {
      // ...
      public function __clone() {
         $this->eletkor += 20;
      }
      // ...
   }

   $karoly = new Ember("Károly", 20, "gyártulajdonos");
   $azIdosebbKaroly = clone $karoly;      // __clone() metódus hívása

   $karoly->bemutatkozik();
   $azIdosebbKaroly->bemutatkozik();
?>
```

```
Szia! Károly vagyok, 20 éves.
Szia! Károly vagyok, 40 éves.
```

C. Konstans és statikus adattagok

- **Konstans adattagok**at a `const` kulcsszóval hozhatunk létre egy osztályon belül
  - nevük elé nem írunk `$`-et
  - öröklődéssel a konstansok felüldefiniálhatók
  - a konstans adattagok elérése:
    - osztályon belül: `self::KONSTANS_NEVE`
    - osztályon kívül: `OsztalyNeve::KONSTANS_NEVE`

```php
<?php
   class Kor {
      private $r;              // a kör sugara
      const PI = 3.14;         // a pí értékét konstansként adjuk meg

      public function kerulet() { return 2 * $this->r * self::PI; }
   }
?>
```

- **Statikus adattagok** létrehozására a `static` kulcsszó szolgál
  - egyetlen példányban létezik, az osztály minden példánya osztozik rajta
  - tehát lényegében nem az objektumhoz, hanem az osztályhoz tartozik
  - a statikus adattagok szabályos elérése:
    - osztályon belül: `self::$adattagNeve`
    - osztályon kívül: `OsztályNeve::$adattagNeve`
  - nem csak adattagok, hanem metódusok is lehetnek statikusak

```php
<?php
   class Ember {
      private $nev;
      private $eletkor;
      protected $foglalkozas;
      public static $emberekSzama;        // statikus adattag

      public function __construct($nev, $eletkor, $foglalkozas) {
         $this->nev = $nev;
         $this->eletkor = $eletkor;
         $this->foglalkozas = $foglalkozas;
         self::$emberekSzama++; // osztályon belüli hivatkozás a statikus adattagra
      }

      public function __destruct() {
         self::$emberekSzama--; // osztályon belüli hivatkozás a statikus adattagra
      }

      // ...
   }

   $ember1 = new Ember("Dávid", 21, "kereskedő");
   $ember2 = new Ember("Az öreg Kovács", 60, "macskatulajdonos");
   $ember3 = new Ember("Dante", 26, "irodalmár");

   echo Ember::$emberekSzama . "<br/>";     // osztályon kívüli hivatkozás… (3)
   echo $ember2::$emberekSzama . "<br/>";   // ez is valid (3)
?>
```

D. Öröklődés

- Lényege: egy osztálynak (ősosztály) egy speciális változatát (gyermekosztály) készítjük el
  - a gyermek megörökli az ős összes adattagját és metódusát
  - a gyermek az örökölt adattagok és metódusok mellett újakat is tartalmazhat
  - a gyermekben felül lehet definiálni egy-egy örökölt metódus működését (**overriding**)
- Szintaxis: `class GyermekOsztaly extends OsOsztaly { … }`
  - **csak egyszeres öröklődés!** – a gyermekosztály max. 1 ősosztályból származhat
- A gyermekosztályban meghívhatjuk az ősosztály metódusait: `parent::metodusNeve()`

Példa: Hozzunk létre egy `Hallgato` osztályt, ami az előbbi `Ember` osztálynak lesz gyermeke.

```php
<?php
   class Ember {
      // ezt már megírtuk… :)
      public function getNev() { return $this->nev; }
   }

   class Hallgato extends Ember {
      private $neptun;        // egy új adattag (az örökölteken kívül)

      public function __construct($nev, $eletkor, $neptun) {
         parent::__construct($nev, $eletkor, "hallgató");  // ős konstruktor hívása
         $this->neptun = $neptun;
      }

      public function bemutatkozik() {  // felülírjuk az ősben lévő metódust
         echo parent::getNev() . " vagyok, egyetemi hallgató. <br/> ";
      }

      public function osztondij($atlag) {  // egy új metódus (az örökölteken kívül)
         if ($atlag > 3.0) {
            echo parent::getNev() . " (" . $this->neptun . ") ösztöndíjat kap. <br/>";
         } else {
            echo parent::getNev() . " nem kap ösztöndíjat. <br/>";
         }
      }
   }

   $ember1 = new Ember("Margit", 72, "nyugdíjas");
   $ember2 = new Hallgato("Ricsi", 21, "NEP4LF");
   $ember3 = new Hallgato("Tamás", 24, "ASD123");

   $ember1->bemutatkozik();
   $ember2->bemutatkozik();
   $ember2->osztondij(2.8);
   $ember3->osztondij(4.85);
?>
```

```
Szia! Margit vagyok, 72 éves.
Ricsi vagyok, egyetemi hallgató.
Ricsi nem kap ösztöndíjat.
Tamás (ASD123) ösztöndíjat kap.
```

E. Típusellenőrzés

- `$obj` **`instanceof`** `Oszt`: visszaadja, hogy a `$obj` objektum az `Oszt` osztály példánya-e
- Más típusok ellenőrzése az **`is_`**`típus()` függvényekkel lehetséges (lásd: 7. jegyzet)

```php
<?php
   class Ember { /* ... */ }
   class Hallgato extends Ember { /* ... */ }

   $e1 = new Ember("Noémi", 21, "papagájidomár");
   $e2 = new Hallgato("Béla", 30, "F00B4R");

   var_dump($e1 instanceof Ember);        // bool(true)
   var_dump($e2 instanceof Ember);        // bool(true)
?>
```

F. A `final` kulcsszó

- A **`final`** metódusokat nem lehet felüldefiniálni a gyermekosztályban
- A **`final`** osztályokból nem öröklődhet másik osztály

```php
<?php
   class MyClass { final public function myFinalMethod() { /* … */ }

   final class MyFinalClass { /* … */ }
?>
```

G. Az `abstract` kulcsszó

- Az **`abstract`** kulcsszóval absztrakt metódusokat, illetve absztrakt osztályokat készíthetünk
- **Absztrakt metódusok**
  - olyan metódus az osztályban, amelynek nincsen törzse
  - pl. mivel egy állatról általánosságban nem tudjuk, hogy milyen hangot ad, ezért az `Allat` osztályon belül a `hangotAd()` metódust absztrakttá tehetjük
- **Absztrakt osztályok**
  - ha egy osztály tartalmaz legalább 1 absztrakt metódust, akkor az osztály kötelezően absztrakt kell, hogy legyen
  - az absztrakt osztály nem példányosítható
  - a gyermekosztályban az összes absztrakt metódust felül kell definiálni
    - ellenkező esetben a gyermekosztálynak is absztraktnak kell lennie

H. Interface-ek

- Létrehozás: **interface** kulcsszóval
- Csak absztrakt metódusokat és konstans adattagokat tartalmaz
- Az interfészeket az **implements** kulcsszóval valósíthatjuk meg osztályok esetén
  - az interface absztrakt metódusainál nem kell kiírni az **abstract** kulcsszót
  - az implementáló osztály az interface összes absztrakt metódusát meg kell, hogy valósítsa (egyébként absztrakttá tesszük az osztályt)
  - egy osztály több interface-t is implementálhat
    - ezt használhatjuk a többszörös öröklődés hiányának megoldására

MEGJEGYZÉS

A többszörös öröklődés hiányára hatékonyabb megoldást nyújtanak a **trait**-ek. Róluk ezen a linken olvashatsz (illetve előadáson is előkerülnek).

```php
<?php
   abstract class Allat {     // absztrakt osztály
      protected $nev;

      public function __construct($nev) { $this->nev = $nev; }
      abstract public function hangotAd();  // absztrakt metódus
   }

   interface Negylabu {
      const LABAK_SZAMA = 4;
   }

   interface Ragadozo {
      public function vadaszik();
      public function eszik($mit);
   }

   class Macska extends Allat implements Negylabu, Ragadozo {
      // megvalósítjuk az absztrakt ős és az interface-ek absztrakt metódusait
      public function hangotAd() { echo "Miaú! <br/>"; }
      public function vadaszik() { echo $this->nev . " vadászik… <br/>"; }
      public function eszik($mit) { echo $this-> nev . " megette: " . $mit . "<br/>"; }
   }

   // $allat1 = new Allat("Blöki");       // absztrakt osztály nem példányosítható
   $allat2 = new Macska("Garfield");

   $allat2->hangotAd();          // "Miaú!"
   $allat2->vadaszik();          // "Garfield vadászik… "
   $allat2->eszik("csirke");     // "Garfield megette: csirke"
?>
```

I. Névterek

- Létrehozás: **namespace** kulcsszóval
- Osztályok, interface-ek, függvények, konstansok egységbe szervezése
  - cél: logikai csoportosítás, névütközések elkerülése
  - kb. mint Javában a package-ek

```php
<?php
   namespace QuickMaffs {
      const PI = 3.14;
      class ComplexNumber { /* ... */ }
      function add($a, $b) { return $a + $b; }
   }
?>
```

4. Kivételkezelés

- **Kivétel** (`Exception`): a program végrehajtása során bekövetkező rendkívüli esemény, amely meggátolja a program futását
  - a **throw** utasítással mi is dobhatunk kivételt
  - ha valahol el lett dobva egy kivétel, akkor azt el is tudjuk kapni
    - **try** { /* a kód azon része, ahol kivétel dobódhat */ }
    - **catch** (Exception $e) { /* Exception típusú kivétel elkapása */ }
    - **finally** { /* mindig lefutó kódrész */ }

```php
<?php
   function osztas(int $a, int $b) {
      if ($b == 0)      // ha az osztó nulla, dobunk egy kivételt
         throw new Exception("HIBA: Nullával való osztás!");

      return $a / $b;
   }

   try {
      echo osztas(5, 2) . "<br/>";
      echo osztas(5, 0) . "<br/>"; // kivételt eredményez -> catch ág
   } catch (Exception $e) {
      echo $e->getMessage() . "<br/>";  // kivétel üzenetének kiírása
   } finally {
      echo "--- kivételkezelés vége --- <br/>";
   }
?>
```

```
2.5
HIBA: Nullával osztás!
--- kivételkezelés vége ---
```