8. ÓRA

Objektumok

Az objektumközpontú programozás veszélyes. Megváltoztatja a programozásról való gondolkodásmódunkat és ha egyszer a hatalmába kerít, nem tudunk tőle szabadulni. A PHP a Perlhöz hasonlóan fokozatosan építette be az objektumközpontúságot a nyelvtanba. A PHP 4-es változatának megjelenésével képessé váltunk arra, hogy programjainkat teljes mértékben objektumközpontú szemléletmódban írjuk.

Ebben az órában a PHP 4-es változatának objektumközpontú tulajdonságaival foglalkozunk és azokat valós életből vett példákra alkalmazzuk. Az órában a következőket tanuljuk meg:

- Mik az osztályok és objektumok?
- Hogyan lehet osztályokat és objektumpéldányokat létrehozni?
- Hogyan lehet tagfüggvényeket és tulajdonságokat létrehozni és elérni?
- Hogyan lehet olyan osztályt létrehozni, mely más osztály leszármazottja?
- Miért jó az objektumközpontú programozás és hogyan segíti munkánk szervezését?

Mit nevezünk objektumnak?

Az objektum változók és függvények egybezárt csomagja, amely egy különleges sablonból jön létre. Az objektum belső működése nagy részét elrejti az objektumot használó elől, úgy, hogy egyszerű hozzáférési felületet biztosít, melyen keresztül kéréseket küldhetünk és információt kaphatunk. A felület különleges függvényekből, úgynevezett tagfüggvényekből (metódusokból) áll. A tagfüggvények mindegyike hozzáférhet bizonyos változókhoz, amelyeket tulajdonságoknak nevezünk.

A típus jellegzetességeit objektumtípusok (osztályok, class) létrehozásával határozzuk meg. Objektumpéldányok (vagy röviden objektumok) létrehozásával pedig egyedeket hozunk létre. Ezen egyedek rendelkeznek a típusuknak megfelelő jellemzőkkel, de természetesen a jellemzők értéke más és más lehet. Például ha létrehozunk egy `gepkocsi` osztályt, akkor az osztálynak lehet egy jellemzője, amelynek a `szin` nevet adjuk. Az objektumpéldányokban a `szin` tulajdonság értéke lehet például `"kék"` vagy `"zöld"`.

Az osztály tagfüggvények és tulajdonságok együttese. Az osztályokat a `class` kulcsszó segítségével hozhatjuk létre. Az osztályok sablonok, amelyekből objektumokat állíthatunk elő.

ÚJDONSÁG Az objektum az osztály egy példánya, vagyis egy objektum nem más, mint az osztályban rögzített működési szabályok megtestesülése. Egy adott típusú objektum a `new` kulcsszó után írt objektumtípus nevének segítségével testesíthető meg. Amikor egy objektumpéldány létrejön, összes tulajdonsága és tagfüggvénye elérhetővé válik.

Az objektumközpontú program legnagyobb előnye valószínűleg a kód újrahasznosíthatósága. Mivel az osztályokat egységbezárt objektumok létrehozására használjuk, egyszerűen vihetők át egyik projektből a másikba, ráadásul lehetőség van arra is, hogy gyermekosztályokat hozzunk létre, amelyek a szülő tulajdonságait öröklik és képesek azt kiegészíteni vagy módosítani. E módszer segítségével rengeteg összetett vagy egyedi tulajdonsággal rendelkező objektumot, vagyis objektumcsaládot lehet létrehozni, amelyek egy alaposztályra épülnek, de közben egyéni tulajdonságokkal is rendelkeznek, egyedi tagfüggvényeket is biztosítanak.

Az objektumközpontú programozás bemutatásának talán legjobb módja, ha a bemutatott példák alapján kísérletezgetünk.

Objektum létrehozása

Ahhoz, hogy létre tudjunk hozni egy objektumot, először létre kell hozni a sablont, amelyre épül. Ezt a sablont hívjuk osztálynak. A PHP 4-es változatában ilyen sablont a `class` kulcsszóval hozhatunk létre:

```
class elso_osztaly
     {
     // ez egy nagyon buta osztály
     }
```

Az `elso_osztaly` a minta, amely alapján tetszőleges számú `elso_osztaly` típusú objektumot hozhatunk létre. Objektumot létrehozni a `new` kulcsszó segítségével lehet:

```
$obj1 = new elso_osztaly();
$obj2 = new elso_osztaly();
print "\$obj1 " . gettype($obj1) . " típusú<br>";
print "\$obj2 " . gettype($obj2) . " típusú<br>";
```

A PHP `gettype()` függvényével ellenőriztük, hogy az $obj1 és az `$obj2` változókban valóban objektum van-e. A `gettype()` számára egy tetszőleges típusú változót kell átadnunk, a függvény pedig egy karakterlánccal tér vissza, ami a változó típusát tartalmazza. Az előző programrészletben a `gettype()` visszatérési értéke `"object"`, amit a `print` segítségével a böngésző számára kiírtunk.

Meggyőződhettünk róla, hogy két objektumunk van. A példának első ránézésre túl sok hasznát még nem látjuk, de segítségével újabb nézőpontból vizsgálhatjuk az osztályokat. Az osztályokra úgy tekinthetünk, mint öntőmintára: annyi objektumot önthetünk a segítségével, amennyit csak szeretnénk. Adjunk néhány további szolgáltatást az osztályhoz, hogy objektumaink kicsit érdekesebbek legyenek.

Objektumtulajdonságok

Az objektumok a tulajdonság néven említett különleges változók révén működnek. Ezek az osztályban bárhol létrehozhatók, de az átláthatóság kedvéért az osztálymeghatározás elejére célszerű kigyűjteni azokat, így mi is ezt tesszük. A tulajdonságok lehetnek értékek, tömbök, de más objektumok is:

```
class elso_osztaly
     {
     var $nev = "Gizike";
     }
```

Figyeljük meg, hogy a változót a `var` kulcsszó segítségével hoztuk létre; ha ezt egy osztályon belül nem írjuk ki, értelmezési hibát (parse error) kapunk. Most, hogy ezt a változót létrehoztuk, minden `elso_osztaly` típusú objektumnak lesz egy `nev` tulajdonsága, melynek `"Gizike"` lesz a kezdeti értéke. Ehhez a tulajdonsághoz az objektumon kívülről is hozzá lehet férni, sőt meg is lehet változtatni:

```
class elso_osztaly
     {
     var $nev = "Gizike";
     }
$obj1 = new elso_osztaly();
$obj2 = new elso_osztaly();
$obj1->nev = "Bonifác";
print "$obj1->nev<br>";
print "$obj1->nev<br>";
```

A `->` műveletjel teszi lehetővé, hogy egy objektum tulajdonságait elérhessük vagy módosíthassuk. Bár az `$obj1` és az `$obj2` objektumokat `"Gizike"` névvel hoztuk létre, rábírtuk az `$obj2` objektumot, hogy meggondolja magát, a nev tulajdonsághoz a `"Bonifác"` értéket rendelve, mielőtt a `->` jelet ismét használva kiítuk volna mindkét objektum `nev` tulajdonságának értékét.

Az objektumközpontú nyelvek, mint a Java, megkívánják, hogy a programozó beállítsa a tulajdonságok és tagfüggvények biztonsági szintjét. Ez azt jelenti, hogy a hozzáférés úgy korlátozható, hogy csak bizonyos, az objektumot kezelő magas szintű függvények legyenek elérhetők, a belső használatra szánt tulajdonságok, tagfüggvények pedig biztonságosan elrejthetők. A PHP-nek nincs ilyen védelmi rendszere. Az objektum összes tulajdonsága elérhető, ami problémákat okozhat, ha a tulajdonságot (kívülről) nem lenne szabad megváltoztatni.

Az objektumot adattárolásra is használhatjuk, de az objektumok többre képesek, mint egy egyszerű asszociatív tömb utánzása. A következő részben áttekintjük az objektumok tagfüggvényeit, amelyek segítségével objektumaink életre kelnek.

Az objektumok tagfüggvényei

A tagfüggvény olyan függvény, amelyet egy osztályon belül hozunk létre. Minden objektumpéldány képes meghívni osztálya tagfüggvényeit. A 8.1. példaprogram az `elso_osztaly` nevű osztályt egy tagfüggvénnyel egészíti ki.

8.1. program Egy osztály tagfüggvénnyel

```
 1: <html>
 2: <head>
 3: <title>8.1. program Egy osztály
    tagfüggvénnyel</title>
 4: <body>
 5: <?php
 6: class elso_osztaly
 7:      {
 8:      var $nev;
 9:      function koszon()
10:          {
11:          print "Üdvözlöm!";
12:          }
13:      }
14: $obj1 = new elso_osztaly();
15: $obj1->koszon(); // kiírja, hogy "Üdvözlöm!"
16: ?>
17: </body>
18: </html>
```

Amint látjuk, a tagfüggvény a szokványos függvényekhez nagyon hasonlóan viselkedik. A különbség csak annyi, hogy a tagfüggvényt mindig osztályon belül hozzuk létre. Az objektumok tagfüggvényeit a -> műveletjel segítségével hívhatjuk meg. Fontos, hogy a tagfüggvények hozzáférnek az objektum változóihoz. Már láttuk, hogyan lehet egy objektumtulajdonságot az objektumon kívülről elérni, de hogyan hivatkozhat egy objektum saját magára? Lássuk a 8.2. példaprogramot:

8.2. program Tulajdonság elérése tagfüggvényből

```
 1: <html>
 2: <head>
 3: <title>8.2. program Tulajdonság elérése
           tagfüggvényből</title>
 4: <body>
 5: <?php
 6: class elso_osztaly
 7:      {
 8:      var $nev = "Krisztián";
 9:      function koszon()
```

8.2. program (folytatás)

```
10:          {
11:          print"Üdvözlöm! $this->nev vagyok.<BR>";
12:          }
13:      }
14:
15: $obj1 = new elso_osztaly();
16: $obj1->koszon(); // kiírja, hogy "Üdvözlöm! Krisztián
                                          vagyok."
17: ?>
18: </body>
19: </html>
```

Az osztálynak van egy különleges változója, a `$this`, ami az objektumpéldányra mutat. Tekinthetjük személyes névmásnak. Bár programunkban az objektumra hivatkozhatunk azzal a változóval, amihez hozzárendeltük (például `$obj1`), az objektumnak hivatkoznia kell saját magára, erre jó a `$this` változó. A `$this` változó és a `->` műveletjel segítségével az osztályon belülről az osztály minden tulajdonságát és tagfüggvényét elérhetjük.

Képzeljük el, hogy minden egyes `elso_osztaly` típusú objektumhoz más és más `nev` tulajdonságot szeretnénk rendelni. Ezt megtehetnénk „kézzel”, az objektum `nev` tulajdonságát megváltoztatva, de létrehozhatunk egy tagfüggvényt is, ami ezt teszi. Az utóbbi megoldást láthatjuk a 8.3. példaprogramban.

8.3. program Tulajdonság értékének módosítása tagfüggvényen belülről

```
 1: <html>
 2: <head>
 3: <title>8.3. program Tulajdonság értékének módosítása
           tagfüggvényen belülről</title>
 4: </head>
 5: <body>
 6: <?php
 7: class elso_osztaly
 8:      {
 9:      var $nev = "Krisztián";
10:      function atkeresztel( $n )
11:          {
12:          $this->nev = $n;
13:          }
```

8.3. program (folytatás)

```
14:      function koszon()
15:          {
16:          print"Üdvözlöm! $this->nev vagyok.<BR>";
17:          }
18:      }
19:
20: $obj1 = new elso_osztaly();
21: $obj1->atkeresztel("Aladár");
22: $obj1->koszon(); // kiírja, hogy "Üdvözlöm! Aladár
                                        vagyok."
23: ?>
24: </body>
25: </html>
```

Az objektum `nev` tulajdonsága a létrehozáskor még `"Krisztián"` volt, de aztán meghívtuk az objektum `atkeresztel()` tagfüggvényét, ami `"Aladár"` értékre változtatta azt. Láthatjuk, hogy az objektum képes saját tulajdonságait módosítani. Figyeljük meg, hogy a tagfüggvénynek a hagyományos függvényekhez hasonlóan adhatunk át paramétereket.

Még mindig van egy fogás, amit nem ismertünk meg. Ha egy metódusnak éppen azt a nevet adjuk, mint az osztálynak (az előző példában ez az `elso_osztaly`), akkor a metódus automatikusan meghívódik, amikor egy új objektumpéldány létrejön. E módszer segítségével már születésükkor testreszabhatjuk objektumainkat. A létrejövő példány az ezen függvénynek átadott paraméterek vagy más tényezők alapján hozhatja magát alapállapotba. Ezeket a különleges metódusokat konstruktoroknak (létrehozó függvényeknek) hívjuk. A 8.4. példaprogramban az `elso_osztaly` objektumtípus konstruktorral kiegészített változatát láthatjuk.

8.4. program Konstruktorral rendelkező osztály

```
1: <html>
2: <head>
3: <title>8.4. program Konstruktorral rendelkező
   osztály</title>
4: </head>
5: <body>
6: <?php
```

8.4. program (folytatás)

```
 7: class elso_osztaly
 8:      {
 9:      var $nev;
10:      function elso_osztaly( $n="Anonymous" )
11:          {
12:          $this->nev = $n;
13:          }
14:      function koszon()
15:          {
16:          print"Üdvözlöm! $this->nev vagyok.<BR>";
17:          }
18:      }
19:
20: $obj1 = new elso_osztaly("Krisztián");
21: $obj2 = new elso_osztaly("Aladár");
22: $obj1->koszon(); // kiírja, hogy "Üdvözlöm! Krisztián
                                      vagyok."
23: $obj2->koszon(); // kiírja, hogy "Üdvözlöm! Aladár
                                      vagyok."
24: ?>
25: </body>
26: </html>
```

Amikor egy `elso_osztaly` típusú objektumot létrehozunk, az `elso_osztaly()` konstruktor automatikusan meghívásra kerül. Ha az objektum létrehozásakor nem adunk meg nevet, a paraméter értéke `"anonymus"` lesz.

Egy bonyolultabb példa

Most készítsünk együtt egy kicsit bonyolultabb és hasznosabb programot: egy táblázat osztályt hozunk létre, amelyben az egyes oszlopokat nevük alapján is elérhetjük. Az adatszerkezet soralapú lesz, a böngészőben való megjelenítést pedig egy butácska függvényre bízzuk. A táblázat takaros formázása a probléma megoldásának ebben a fázisában nem szükséges.

Az osztály tulajdonságai

Először is el kell döntenünk, hogy milyen tulajdonságokat tároljunk az objektumban. Az oszlopok neveit egy tömbben tároljuk, a sorokat pedig egy kétdimenziós tömbben. A táblázat oszlopainak számát külön, egy egész típusú változóba helyezzük.

```
class Tablazat
     {
     var $tablazatSorok = array();
     var $oszlopNevek = array();
     var $oszlopszam;
     }
```

A konstruktor

A táblázat létrehozásakor a programnak már tudnia kell, mik lesznek a feldolgozandó oszlopok nevei. Ez az információ az objektumhoz a konstruktor egy karakterlánc típusú tömbparamétere segítségével fog eljutni. Ha az oszlopok neveit a konstruktor már ismeri, könnyűszerrel meghatározhatja az oszlopok számát és beírhatja az `$oszlopszam` tulajdonságba:

```
function Tablazat( $oszlopNevek )
      {
      $this->oszlopNevek = $oszlopNevek;
      $this->oszlopszam = count ( $oszlopNevek );
      }
```

Ha feltesszük, hogy a konstruktor számára helyes információt adtunk át, az objektum már tudni fogja oszlopainak számát és nevét. Mivel ez az információ tulajdonságokban (mezőkben) tárolódik, minden tagfüggvény számára hozzáférhető.

Az ujSor() tagfüggvény

A `Tablazat` osztály a sorokat tömbök formájában kapja meg. Ez persze csak akkor működik helyesen, ha a tömbben és a fejlécben az oszlopok sorrendje azonos:

```
function ujSor( $sor )
     {
     if ( count ($sor) != $this->oszlopszam )
          return false;
     array_push($this->tablazatSorok, $sor);
     return true;
     }
```

Az `ujSor()` tagfüggvény egy tömböt vár, amit a `$sor` nevű változóban kap meg. A táblázat oszlopainak számát már az `$oszlopszam` tulajdonságba helyeztük, így most a `count()` függvénnyel ellenőrizhetjük, hogy a kapott `$sor` paraméter megfelelő számú oszlopot tartalmaz-e. Ha nem, akkor a függvény `false` értéket ad vissza.

A PHP 4-es változatának `array_push()` függvénye segítségével az új sort a `$tablazatSor` tömb végéhez fűzhetjük. Az `array_push()` két paramétert vár: a tömböt, amihez az új elemet hozzá kell adni, illetve a hozzáadandó elemet. Ha a második paraméter maga is egy tömb, akkor az egy elemként adódik az első tömbhöz, így többdimenziós tömb jön létre. Ezen eljárás segítségével tehát többdimenziós tömböket is létrehozhatunk.

Az ujNevesSor() tagfüggvény

Az `ujSor()` tagfüggvény csak akkor használható kényelmesen, ha a tömbként átadott paraméterben az elemek sorrendje megfelelő. Az `ujNevesSor()` tagfüggvény ennél több szabadságot ad. A metódus egy asszociatív tömböt vár, ahol az egyes elemek kulcsa az oszlopok neve. Ha olyan kulcsú elem kerül a paraméterbe, amely a Tablazat objektum `$oszlopNevek` tulajdonságában nem szerepel, az adott elem egyszerűen nem kerül bele a táblázatba. Ha egy oszlopnév nem jelenik meg a paraméterben kulcsként, akkor az adott oszlopba egy üres karakterlánc kerül.

```
function ujNevesSor( $asszoc_sor )
     {
     if ( count ($asszoc_sor) != $this->oszlopszam )
           return false;
        $sor = array();
     foreach ( $this->oszlopNevek as $oszlopNev )
          {
          if ( ! isset( $asszoc_sor[$oszlopNev] ))
               $asszoc_sor[$oszlopNev] = "";
          $sor[] = $asszoc_sor[$oszlopNev];
          }
     array_push($this->tablazatSorok, $sor);
     }
```

Az `ujNevesSor()`-nak átadott asszociatív tömb az `$asszoc_sor` nevű változóba kerül. Először létrehozunk egy üres `$sor` tömböt, amely majd a táblázatba beillesztendő sort tartalmazza, az elemek helyes sorrendjével. Majd végigvesszük az `$oszlopNevek` tömb elemeit, és megnézzük, hogy a tömbben kaptunk-e ilyen kulcsú elemet az `$asszoc_sor` paraméterben. Ehhez a PHP 4-es `isset()` függvényét használjuk, amely egy változót vár. Ha a változóhoz már rendeltünk értéket, a függvény visszatérési értéke `true`, egyébként `false` lesz. A függvénynek az `$asszoc_sor` tömb azon elemét kell átadnunk, melynek kulcsa megegyezik a következő oszlop nevével. Ha nem létezik ilyen elem az `$asszoc_sor` tömbben, létrehozunk egy ilyet és üres karakterláncot írunk bele. Most már folytathatjuk a végleges `$sor` tömb felépítését és hozzáadhatjuk az `$asszoc_sor` megfelelő elemét, hiszen épp az imént biztosítottuk, hogy az elem létezik. Mire befejezzük

az $oszlopNevek tömb bejárását, a $sor tömb már tartalmazni fogja az $asszoc_sor tömbben átadott elemeket, mégpedig a megfelelő sorrendben, a nem meghatározott értékeket üres karakterláncokkal kitöltve.

Most már van két tagfüggvényünk, amelyek segítségével sorokat adhatunk a Tablazat típusú objektumok $tablazatSorok nevű mezőjéhez. Erről azonban jó lenne valahogy meggyőződni: hiányzik még egy olyan tagfüggvény, amely képes megjeleníteni a táblázatot.

A kiir() tagfüggvény

A kiir() tagfüggvény kiírja a böngészőbe a táblázat fejlécét és a $tablazatSorok tömböt. A függvény csak nyomkövetési célokat szolgál. Az óra későbbi részében egy sokkal szebb megoldást fogunk látni.

```
function kiir()
     {
     print "<pre>";
     foreach ( $this->oszlopNevek as $oszlopNev )
              print "<B>$oszlopNev</B>   ";
     print "\n";
     foreach ( $this->tablazatSorok as $y )
          {
          foreach ( $y as $xcella )
                   print "$xcella   ";
          print "\n";
          }
     print "</pre>";
     }
```

A program magáért beszél. Először az $oszlopNevek elemeit írjuk ki, majd a $tablazatSorok sorait. Mivel a $tablazatSorok tulajdonság kétdimenziós tömb, vagyis a tömb minden eleme maga is egy tömb, kettős ciklust kell használnunk.

Összeáll a kép

A 8.5. példaprogram eddigi munkánk gyümölcsét, a Tablazat osztályt tartalmazza, majd a program végén létrehoz egy Tablazat típusú objektumot és meghívja valamennyi tagfüggvényét.

8.5. program A Tablazat osztály

```
 1: <html>
 2: <head>
 3: <title>8.5. program A Tablazat osztály</title>
 4: </head>
 5: <body>
 6: <?php
 7: class Tablazat
 8:     {
 9:     var $tablazatSorok = array();
10:     var $oszlopNevek = array();
11:     var $oszlopszam;
12:     function Tablazat( $oszlopNevek )
13:         {
14:         $this->oszlopNevek = $oszlopNevek;
15:         $this->oszlopszam = count ( $oszlopNevek );
16:         }
17:
18:     function ujSor( $sor )
19:         {
20:         if ( count ($sor) != $this->oszlopszam )
21:             return false;
22:         array_push($this->tablazatSorok, $sor);
23:         return true;
24:         }
25:
26:     function ujNevesSor( $asszoc_sor )
27:      {
28:      if ( count ($asszoc_sor) != $this->oszlopszam )
29:          return false;
30:      $sor = array();
31:      foreach ( $this->oszlopNevek as $oszlopNev )
32:          {
33:          if ( ! isset( $asszoc_sor[$oszlopNev] ))
34:              $asszoc_sor[$oszlopNev] = "";
35:          $sor[] = $asszoc_sor[$oszlopNev];
36:          }
37:      array_push($this->tablazatSorok, $sor);
38:      }
39:
```

8.5. program (folytatás)

```
40:      function kiir()
41:          {
42:          print "<pre>";
43:          foreach ( $this->oszlopNevek as $oszlopNev )
44:              print "<B>$oszlopNev</B>  ";
45:          print "\n";
46:          foreach ( $this->tablazatSorok as $y )
47:              {
48:              foreach ( $y as $xcella )
49:                  print "$xcella  ";
50:              print "\n";
51:              }
52:          print "</pre>";
53:          }
54:      }
55:
56: $proba = new Tablazat( array("a","b","c"));
57: $proba->ujSor( array(1,2,3));
58: $proba->ujSor( array(4,5,6));
59: $proba->ujNevesSor( array ( "b"=>0, "a"=>6, "c"=>3 ));
60: $proba->kiir();
61: ?>
62: </body>
63: </html>
```

A 8.5. példaprogram kimenetét a 8.1. ábrán láthatjuk.

8.1. ábra

Egy Tablazat típusú objektum kimenete

A kimenet csak akkor jelenik meg helyesen, ha az egy oszlopban levő elemek egyforma hosszúak.

Ami még hiányzik...

Bár az osztály hatékonyan elvégzi a munkát, amit neki szántunk, ha több időnk és helyünk lenne, kibővíthetnénk néhány további szolgáltatással és adatvédelmi lehetőséggel.

Mivel a PHP gyengén típusos nyelv, a mi felelősségünk megbizonyosodni arról, hogy a tagfüggvényeknek átadott paraméterek a megfelelő típusúak-e. Ehhez a tizenhatodik, az adatkezelésről szóló fejezetben tárgyalt függvényeket használhatjuk. Ezen kívül írhatnánk tagfüggvényeket a táblázat bármely oszlop szerinti rendezésére is.

Miért használjunk osztályt?

Miért jobb osztályt használni erre a célra ahelyett, hogy egyszerűen írnánk néhány tömbkezelő függvényt? Vagy miért ne írhatnánk bele egyszerűen a programba a tömbkezelést? Azért, mert ez nem lenne hatékony, mert így az elvont adatok tárolását végző programokba valami egészen más is bekerülne. De vajon miért okoz ez nekünk problémát?

Először is a kódot újra fel lehet használni. Ennek célja jól meghatározott: bizonyos általános adatkezelést tesz lehetővé, amelyet ezután beilleszthetünk bármely programba, melynek arra van szüksége, hogy ilyen típusú adatokat kezeljen és kiírjon.

Másodszor, a `Tablazat` osztály tevékeny: megkérhetjük arra, hogy adatokat írjon ki, anélkül, hogy bajlódnunk kellene azzal, hogy végigjárjuk a `$tablazatSorok` elemeit.

Harmadszor, az objektumba egy felületet is beépítettünk. Ha elhatározzuk, hogy később javítunk az osztály működésén, anélkül tehetjük meg, hogy a program többi részét módosítanánk, ha a korábban létrehozott tagfüggvények kívülről ugyanúgy viselkednek.

Végül, létrehozhatunk újabb osztályokat is, amelyek már korábban létrehozott osztályoktól örökölnek, kiterjeszthetik vagy módosíthatják a már meglévő tagfüggvényeket. Ez teszi igazán hatékonnyá az objektumközpontú programozást, melynek kulcsszavai az egységbezárás, az öröklés és a kód-újrahasznosítás.

Öröklés

Ahhoz, hogy olyan osztályt hozzunk létre, amely szolgáltatásait szülőjétől örökli, egy kicsit módosítanunk kell az osztály meghatározását. A 8.6. példaprogram ismét a fejezet első felében levő témakörből mutat egy példát.

8.6. program Másik osztálytól öröklő osztály

```
 1: <html>
 2: <head>
 3: <title>8.6. program Másik osztálytól öröklő osztály
    </title>
 4: </head>
 5: <body>
 6: <?php
 7: class elso_osztaly
 8:      {
 9:      var $nev = "Krisztián";
10:      function elso_osztaly( $n )
11:           {
12:           $this->nev = $n;
13:           }
14:      function koszon()
15:           {
16:           print "Üdvözlöm! $this->nev vagyok.<BR>";
17:           }
18:      }
19:
20: class masodik_osztaly extends elso_osztaly
21:      {
22:
23:      }
24:
25: $proba = new masodik_osztaly("Krisztián fia");
26: $proba->koszon(); // kiírja, hogy "Üdvözlöm! Krisztián
                                        fia vagyok."
27: ?>
28: </body>
29: </html>
```

Figyeljük meg, hogyan származtattunk az `elso_osztaly`-tól egy új osztályt! Az `extends` kulcsszót az osztály meghatározásában kell használnunk. Ezzel azt adjuk meg, hogy a kulcsszó után meghatározott osztály minden tulajdonságát és tagfüggvényét örökölni szeretnénk. Az összes `masodik_osztaly` típusú objektum rendelkezni fog egy `koszon()` nevű tagfüggvénnyel és egy `$nev` nevű tulajdonsággal, mint ahogyan minden `elso_osztaly` típusú objektum is rendelkezik ezekkel.

Ha ez nem lenne elég, nézzük tovább a 8.6. példaprogramot és keressünk szokatlan dolgokat! Például figyeljük meg, hogy még konstruktort sem adtunk meg a `masodik_osztaly` osztálynak. Akkor hogy lehet az, hogy az objektum `$nev` tulajdonsága az alapértelmezett `"Krisztián"` karakterláncról arra a `"Krisztián fia"` karakterláncra változott, amit a konstruktor létrehozásakor átadtunk? Mivel nem hoztunk létre új konstruktort, a szülő objektumtípus (`elso_osztaly`) konstruktora hívódott meg.

Ha egy osztályból egy másikat származtatunk, amelynek nincsen saját konstruktora, akkor a szülő osztály konstruktora hívódik meg, amikor egy gyermekobjektumot hozunk létre. Ez a PHP 4-es változatának újdonsága.

A szülő tagfüggvényeinek felülírása

A `masodik_osztaly` típusú objektumok most pontosan úgy viselkednek, mint az `elso_osztaly` típusúak. Az objektumközpontú világban a gyermek osztályban felül lehet írni a szülő osztály tagfüggvényeit, ezáltal lehetővé válik, hogy a gyermek objektumok bizonyos tagfüggvényei a szülőkétől eltérően viselkedjenek (többalakúság, polimorfizmus), más tagfüggvények viszont érintetlenül hagyva ugyanúgy jelenjenek meg a gyermek osztályban. A 8.7. példában lecseréljük a `masodik_osztaly` `koszon()` tagfüggvényét.

8.7. program Egy felülírt tagfüggvény

```
1: <html>
2: <head>
3: <title>8.7. program Egy felülírt tagfüggvény</title>
4: </head>
5: <body>
6: <?php
```

8.7. program (folytatás)

```
 7: class elso_osztaly
 8:      {
 9:      var $nev = "Krisztián";
10:      function elso_osztaly( $n )
11:           {
12:           $this->nev = $n;
13:           }
14:      function koszon()
15:           {
16:           print "Üdvözlöm! $this->nev vagyok.<BR>";
17:           }
18:      }
19:
20: class masodik_osztaly extends elso_osztaly
21:      {
22:      function koszon()
23:           {
24:           print "Azért se mondom meg a nevem!<BR>";
25:           }
26:      }
27:
28: $proba = new masodik_osztaly("Krisztián fia");
29: $proba->koszon(); // kiírja, hogy "Azért se mondom
                                        meg a nevem!"
30: ?>
31: </body>
32: </html>
```

A gyermek `koszon()` tagfüggvénye hívódik meg, mert előnyt élvez a szülő hasonló nevű tagfüggvényével szemben.

A felülírt tagfüggvény meghívása

Előfordulhat, hogy a szülő osztály szolgáltatásait szeretnénk használni, de újakat is be szeretnénk építeni. Az objektumközpontú programozásban ez megvalósítható. A 8.8. példaprogramban a `masodik_osztaly koszon()` tagfüggvénye meghívja az `elso_osztaly` tagfüggvényét, amelyet éppen felülír.

8.8. program Felülírt tagfüggvény meghívása

```
 1: <html>
 2: <head>
 3: <title>8.8. program Felülírt tagfüggvény
          meghívása</title>
 4: </head>
 5: <body>
 6: <?php
 7: class elso_osztaly
 8:     {
 9:     var $nev = "Krisztián";
10:     function elso_osztaly( $n )
11:         {
12:         $this->nev = $n;
13:         }
14:     function koszon()
15:         {
16:         print "Üdvözlöm! $this->nev vagyok.<BR>";
17:         }
18:     }
19:
20: class masodik_osztaly extends elso_osztaly
21:     {
22:     function koszon()
23:         {
24:         print "Azért se mondom meg a nevem! - ";
25:         elso_osztaly::koszon();
26:         }
27:     }
28:
29: $proba = new masodik_osztaly("Krisztián fia");
30: $proba->koszon(); // kiírja, hogy "Azért se mondom
    meg a nevem! - Üdvözlöm! Krisztián fia vagyok."
31: ?>
32: </body>
33: </html>
```

A

```
szuloOsztalyNeve::tagfuggvenyNeve()
```

sémát használva bármely felülírt tagfüggvényt meghívhatunk. A séma új, a PHP 3-as változatában még hibát okozott volna.

Egy példa az öröklésre

Már láttuk, hogyan tud egy osztály egy másiktól örökölni, annak tagfüggvényeit felülírni és képességeit kibővíteni. Most a tanultaknak hasznát is vehetjük. Készítsünk egy osztályt, amely a 8.5. példaprogramban szereplő `Tablazat` osztálytól örököl! Az új osztály neve `HTMLTablazat` lesz. Ezt az osztály azért szükséges, hogy kiküszöbölje a `Tablazat` `kiir()` tagfüggvénynek szépséghibáit és a böngésző lehetőségeit kihasználva szép kimenetet eredményezzen.

A HTMLTablazat saját tulajdonságai

A `HTMLTablazat` arra szolgál, hogy a már korábban létrehozott `Tablazat` típusú objektumot szabványos HTML formában jeleníthessük meg. Példánkban a táblázat `cellPadding` és a cellák `bgColor` tulajdonságai módosíthatók, a valós programokban azonban természetesen több tulajdonságot szeretnénk majd beállítani.

```
class HTMLTablazat extends Tablazat
     {
     var $hatterSzin;
     var $cellaMargo = 2;
     }
```

Egy új osztályt hoztunk létre, amely a `Tablazat` osztálytól fog örökölni. Két új tulajdonságot adtunk meg, az egyiknek a kezdőértékét is meghatároztuk.

A konstruktor

Már láthattuk, hogy a szülő konstruktora hívódik meg, ha a gyermekosztályban nem hozunk létre konstruktort. Most azonban azt szeretnénk, hogy konstruktorunk többre legyen képes, mint amennyit a Tablazat osztály konstruktora tett, ezért azt felül kell írnunk.

```
function HTMLTablazat( $oszlopNevek, $hatter="#ffffff" )
     {
     Tablazat::Tablazat($oszlopNevek);
     $this->hatterSzin=$hatter;
     }
```

A `HTMLTablazat` paramétereiben az oszlopok neveinek tömbjét, illetve egy karakterláncot vár. A karakterlánc lesz a HTML táblázat `bgColor` tulajdonsága. Megadtunk egy kezdeti értéket is, így ha nem adunk át második paramétert a konstruktornak, a háttér színe fehér lesz. A konstruktor meghívja a `Tablazat` osztály konstruktorát a kapott oszlopnév-tömbbel. A lustaság erény a programozás terén: hagyjuk, hogy a `Tablazat` konstruktora tegye a dolgát és a továbbiakban

nem foglalkozunk a táblázat kezelésével (ha egyszer már megírtuk, akkor használjuk is…). A konstruktor másik dolga a `HTMLTablazat` `hatterSzin` tulajdonságának beállítása.

Ha a gyermek osztály rendelkezik konstruktorral, akkor a szülő osztály konstruktora már nem kerül automatikusan meghívásra. Ezért ha szükséges, a gyermek osztályból kell meghívni a szülő konstruktorát.

A cellaMargoAllit() tagfüggvény

A származtatott osztály saját, új tagfüggvényeket is létrehozhat.
A `cellaMargoAllit()` tagfüggvény segítségével a felhasználó a táblázat szövege és a táblázat közötti üres rés nagyságát állíthatja be. Persze megtehetné ezt a `cellaMargo` tulajdonság közvetlen elérésével is ,de ez nem túl jó ötlet. Alapszabály, hogy az objektumot használó személy érdekében legjobb tagfüggvényeket létrehozni erre a célra. Az osztályok bonyolultabb leszármazottaiban a `cellaMargoAllit()` tagfüggvény esetleg más tulajdonságokat is kénytelen módosítani, hogy megváltoztathassa a margó méretét. Sajnos nincs elegáns mód ennek a kikényszerítésére.

```
function cellaMargoAllit( $margo )
     {
     $this->cellaMargo = $margo;
     }
```

A kiir() tagfüggvény

A `kiir()` tagfüggvény felülírja a `Tablazat` osztály azonos nevű tagfüggvényét. Ez a függvény a szülő osztály logikájával azonos módon írja ki a táblázatot, de a HTML `table` elemét használja a táblázat formázására.

```
function kiir()
     {
     print "<table cellPadding=\"$this->cellaMargo\"
          border=1>\n";
     print "<tr>\n";
     foreach ( $this->oszlopNevek as $oszlopNev )
          print "<th bgColor=\"$this
               ->hatterSzin\”>$oszlopNev</th>”;
     print "</tr>\n";
```

```
    foreach ( $this->tablazatSorok as $sor => $cellak )
        {
        print "<tr>\n";
        foreach ( $cellak as $cella )
            print "<td bgColor=\"$this
                ->hatterSzin\">$cella</td>\n";
        print "</tr>\n";
        }
    print "</table>";
    }
```

Ha a `Tablazat` osztálybeli változat világos, az itteni `kiir()` tagfüggvény is elég áttekinthető kell, hogy legyen. Először az `$oszlopNevek` elemeit írjuk ki, majd a `$tablazatSorok` sorait. A formázást a HTML `table` elemének `cellPadding` és `bgColor` tulajdonságai segítségével szabályozhatjuk.

A Tablazat és a HTMLTablazat osztályok a maguk teljességében

A 8.9. példaprogramban a `Tablazat` és a `HTMLTablazat` példákat egy helyen láthatjuk. Létrehozunk egy `HTMLTablazat` típusú objektumot, megváltoztatjuk `cellaMargo` tulajdonságát, írunk bele néhány sort, majd meghívjuk a `kiir()` tagfüggvényt. A valóságban az adatok minden bizonnyal közvetlenül egy adatbázisból származnának.

8.9. program A Tablazat és a HTMLTablazat osztályok

```
 1: <html>
 2: <head>
 3: <title>8.9. program A Tablazat és a HTMLTablazat
           osztályok</title>
 4: </head>
 5: <body>
 6: <?php
 7: class Tablazat
 8:      {
 9:      var $tablazatSorok = array();
10:      var $oszlopNevek = array();
11:      var $oszlopszam;
```

8.9. program (folytatás)

```
12:     function Tablazat( $oszlopNevek )
13:         {
14:         $this->oszlopNevek = $oszlopNevek;
15:         $this->oszlopszam = count ( $oszlopNevek );
16:         }
17:
18:     function ujSor( $sor )
19:         {
20:         if ( count ($sor) != $this->oszlopszam )
21:             return false;
22:         array_push($this->tablazatSorok, $sor);
23:         return true;
24:         }
25:
26:     function ujNevesSor( $asszoc_sor )
27:      {
28:      if ( count ($asszoc_sor) != $this->oszlopszam )
29:          return false;
30:      $sor = array();
31:      foreach ( $this->oszlopNevek as $oszlopNev )
32:          {
33:          if ( ! isset( $asszoc_sor[$oszlopNev] ))
34:              $asszoc_sor[$oszlopNev] = "";
35:          $sor[] = $asszoc_sor[$oszlopNev];
36:          }
37:      array_push($this->tablazatSorok, $sor);
38:      }
39:
40:     function kiir()
41:         {
42:         print "<pre>";
43:         foreach ( $this->oszlopNevek as $oszlopNev )
44:             print "<B>$oszlopNev</B>  ";
45:         print "\n";
46:         foreach ( $this->tablazatSorok as $y )
47:             {
48:             foreach ( $y as $xcella )
49:                 print "$xcella  ";
50:             print "\n";
51:             }
```

8.9. program (folytatás)

```
52:          print "</pre>";
53:          }
54:      }
55:
56: class HTMLTablazat extends Tablazat
57:      {
58:      var $hatterSzin;
59:      var $cellaMargo = 2;
60:      function HTMLTablazat( $oszlopNevek,
          $hatter="#ffffff" )
61:          {
62:          Tablazat::Tablazat($oszlopNevek);
63:          $this->hatterSzin=$hatter;
64:          }
65:
66:      function cellaMargoAllit( $margo )
67:          {
68:          $this->cellaMargo = $margo;
69:          }
70:      function kiir()
71:       {
72:       print "<table cellPadding=\"$this->cellaMargo\"
              border=1>\n";
73:          print "<tr>\n";
74:          foreach ( $this->oszlopNevek as $oszlopNev )
75:          print "<th bgColor=\"$this->hatterSzin\"
                   >$oszlopNev</th>";
76:  print "</tr>\n";
77:  foreach ( $this->tablazatSorok as $sor => $cellak )
78:              {
79:              print "<tr>\n";
80:              foreach ( $cellak as $cella )
81:              print "<td bgColor=\"$this->hatterSzin\"
                        >$cella</td>\n";
82:              print "</tr>\n";
83:              }
84:          print "</table>";
85:          }
86:      }
87:
```

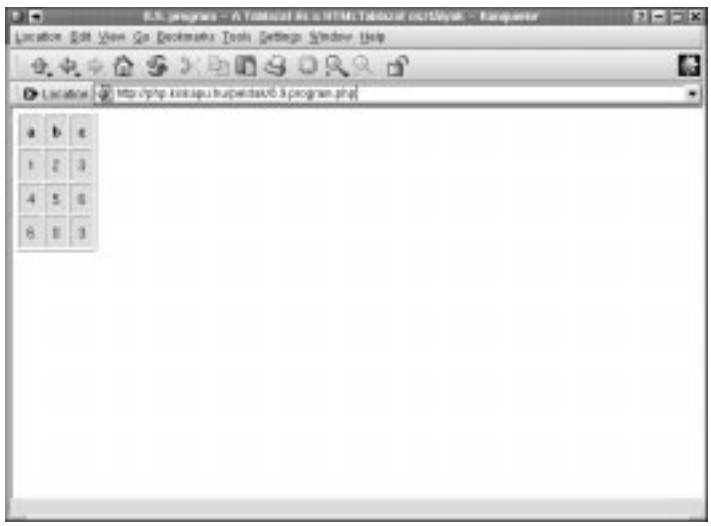
8.9. program (folytatás)

```
88: $proba = new HTMLTablazat( array("a","b","c"),
          "#00FF00");
89: $proba->cellaMargoAllit( 7 );
90: $proba->ujSor( array(1,2,3));
91: $proba->ujSor( array(4,5,6));
92: $proba->ujNevesSor( array ( "b"=>0, "a"=>6, "c"=>3 ));
93: $proba->kiir();
94: ?>
95: </body>
96: </html>
```

A 8.9. példaprogram kimenetét a 8.2. ábrán láthatjuk.

8.2. ábra

Egy HTMLTablazat típusú objektum kimenete



Miért alkalmazzunk öröklést?

Miért vágtuk ketté a feladatot egy `Tablazat` és egy `HTMLTablazat` részre? Biztos, hogy időt és fáradságot takaríthattunk volna meg, ha a HTML táblázat képességeit beépítjük a `Tablazat` osztályba? A válasz kulcsa a rugalmasság kérdése.

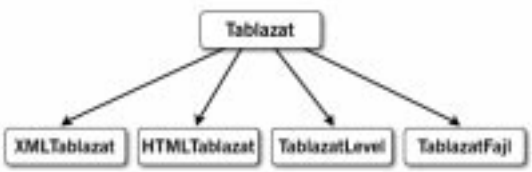
Képzeljük el, hogy egy ügyfél azt a megbízást adja nekünk, hogy hozzunk létre egy osztályt, amely olyan táblázatok kezelésére képes, ahol a táblázat oszlopainak neve van. Ha egyetlen részből álló osztályt hozunk létre, amelybe minden szolgáltatást (a HTML megjelenítés minden apró részletével) beleépítünk, első látásra minden szépnek és jónak tűnhet. De ha visszajön az ügyfél, hogy szeretné, ha a program ízlésesen formázott szöveges kimenetet eredményezne, valószínűleg újabb tagfüggvényeket kellene megpróbálnunk felvenni, melyek megoldják a problémát.

Egy-két héten belül ügyfelünk ráébred, hogy szeretné a programot elektronikus levelek küldésére használni és ha már úgyis fejlesztünk a programon, a cég belső hálózata az XML nyelvet használja, nem lehetne beépíteni a támogatást ehhez is? Ennél a pontnál, ha minden szolgáltatást egyetlen osztályba építünk bele, a program kezd ormótlanul nagy lenni, esetleg a teljes átírását fontolgatjuk.

Játsszuk el ezt a forgatókönyvet a `Tablazat` és a `HTMLTablazat` osztályainkkal! Alaposan sikerült elkülöníteni az adatok feldolgozását és megjelenítését. Ha ügyfelünk azzal a kéréssel fordul hozzánk, hogy szeretné a táblázatot fájlba menteni, csak egy új osztályt kell származtatnunk a `Tablazat` osztályból. Nevezzük ezt `TablazatFajl` osztálynak. Eddigi osztályainkhoz hozzá sem kell nyúlnunk. Ez igaz a `TablazatLevel` és az `XMLTablazat` osztályokra is. A 8.3. ábra az osztályok közötti kapcsolatokat (függőségeket, leszármazásokat) mutatja.

8.3. ábra

Kapcsolatok a Tablazat osztály és gyermekei között



Sőt, tudjuk, hogy minden, a `Tablazat` osztályból származtatott osztálynak van egy `kiir()` tagfüggvénye, így azokat egy tömbbe is gyűjthetjük. Azután végigszaladunk a tömbön, meghívjuk minden elem `kiir()` tagfüggvényét és nem kell azzal foglalkoznunk, hogy az adott elem a `Tablazat` melyik leszármazottja, a megfelelő kiíró függvény kerül meghívásra. Egy egyszerű `Tablazat` osztályból származtatott típusú objektumok tömbje segítségével elektronikus leveleket írhatunk, illetve HTML, XML vagy szöveges állományokat állíthatunk elő a `kiir()` függvény meghívásával.

Összefoglalás

Sajnos nincs rá mód, hogy az objektumközpontú programozást minden szempontból megvizsgáljuk egy rövidke óra alatt, de remélem, hogy a fontosabb lehetőségeket sikerült bemutatni.

Azt, hogy milyen mértékben használunk osztályokat a programokban, nekünk kell eldöntenünk. Az objektumközpontú megközelítést intenzíven használó programok általában némileg több erőforrást igényelnek, mint a hagyományosak, viszont a rugalmasság és átláthatóság jelentősen nő.

Ebben az órában arról tanultunk, hogyan hozhatunk létre osztályokat és belőlük objektumpéldányokat. Megtanultunk tulajdonságokat és tagfüggvényeket létrehozni és elérni. Végül azt is megtanultuk, hogyan hozhatunk létre új osztályokat más osztályokból az öröklés és a tagfüggvény-felülírás segítségével.

Kérdések és válaszok

Ebben az órában néhány barátságtalan fogalommal találkoztunk. Valóban szükséges az objektumközpontúságot megérteni ahhoz, hogy jó PHP programozó válhasson az emberből?
Erre a rövid válasz: nem. A legtöbb PHP programozó alig vagy egyáltalán nem ír objektumközpontú programot. Az objektumközpontú megközelítés nem tesz számunkra olyan dolgokat lehetővé, amelyek eddig elérhetetlenek voltak. A dolog lényege a programok szervezésében, az egyszer megírt programrészek újrahasznosításában és a könnyű fejleszthetőségben rejlik. Még ha el is határoznánk, hogy soha nem fogunk objektumközpontú programot írni, előfordulhat, hogy bele kell javítanunk mások által írt programokba, amelyekben osztályok vannak, ezért értenünk kell a szemlélet alapjait. Ez az óra ebben is segítséget nyújthat.

Nem értem a `$this` változó szerepét.
Egy osztályon belül szükséges az osztály tagfüggvényeit vagy tulajdonságait elérni. Ekkor a `$this` változót kell használni, amely egy mutató az adott objektumra, amelynek valamely tagfüggvényét meghívtuk. Mivel a `$this` változó mutató (hivatkozás), összetevőinek eléréséhez a `->` műveletjelet kell használni.

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

1. Hogyan hoznánk létre egy `uresOsztaly` nevű osztályt, amelynek nincsenek tagfüggvényei és tulajdonságai?

2. Adott egy `uresOsztaly` nevű osztály. Hogyan hoznánk létre egy példányát?

3. Hogyan lehet az osztályba egy tulajdonságot felvenni?

4. Hogyan kell a konstruktor nevét megválasztani?

5. Hogyan lehet egy osztályban konstruktort létrehozni?

6. Hogyan lehet létrehozni közönséges tagfüggvényeket?

7. Hogyan lehet osztályon belülről az osztály tulajdonságaihoz és tagfüggvényeihez hozzáférni?

7. Hogyan lehet osztályon kívülről az osztály tulajdonságaihoz és tagfüggvényeihez hozzáférni?

8. Mit kell tennünk, ha azt szeretnénk, hogy egy osztály más osztálytól örököljön?

Feladatok

1. Hozzuk létre a `Szamolo` nevű osztályt, amely két egész számot tárol. Írjunk hozzá egy `kiszamol()` nevű tagfüggvényt, amely kiírja a két számot a böngészőbe!

2. Hozzuk létre az `Osszead` osztályt, amely a `Szamolo` osztálytól örököl. Írjuk át ennek `kiszamol()` tagfüggvényét úgy, hogy az a két tulajdonság összegét írja ki a böngészőbe!

3. Az előző feladathoz hasonlóan módon készítsük el a `Kivon` osztályt!