

TDK-DOLGOZAT

Készítette: Visnovitz Márton

Objektumorientált PHP library szabványos HTML5 alkalmazások fejlesztéséhez

Tudományos Diákköri Dolgozat

Visnovitz Márton

E-mail: vimtaai@inf.elte.hu

Programtervező informatikus BSc

2. évfolyam

Témavezető: Abonyi-Tóth Andor, műszaki tanár

ELTE IK Média- és Oktatásinformatika Tanszék



Budapest, 2012. november

Tartalomjegyzék

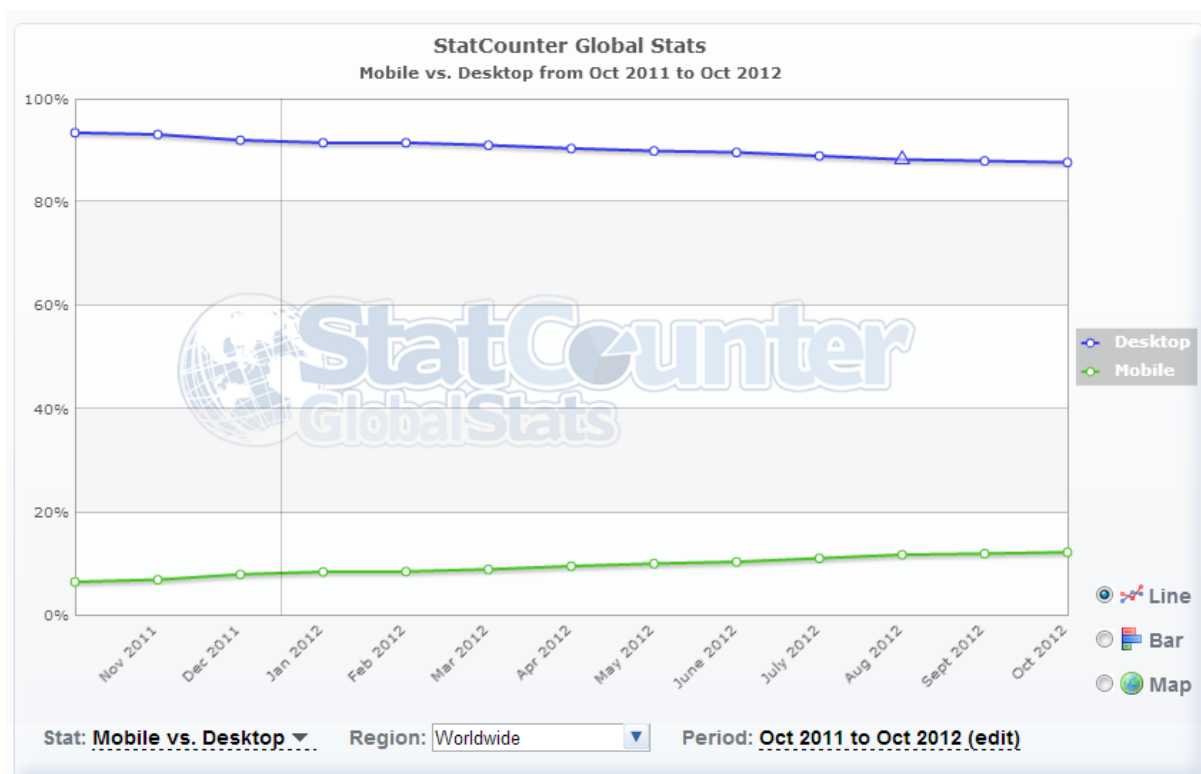
Bevezetés.....	5
A projekt célja	5
Célközönség	6
Hasonló rendszerek	6
A programkönyvtár felépítése.....	9
A <i>Core</i> modul.....	9
A <i>Form</i> modul	9
A <i>Message</i> modul.....	9
Mappa- és fájlstruktúra	10
Osztálystruktúra.....	11
Control	11
Container.....	12
Document.....	13
Bázisosztályok	15
A dokumentum generálása	15
Bővíthetőség	16
Osztálydiagram	17
A library használata	19
Egy példafeladat megoldása	19
A megoldandó feladat.....	19
A példa megoldásának kódja	25
Képernyőkép a megoldásról	27
A kimeneti HTML kód	28
Egyedi megoldások a programban	30
Automatikus konverziók.....	30

Metódusláncolás	31
Fejlesztési lehetőségek	32
A programkönyvtár terjesztése	33
Összefoglalás	34
Eredmény.....	34
Rendszerkövetelmények.....	34
Köszönetnyilvánítás.....	35
Irodalomjegyzék	36
Ábrajegyzék	37
Melléklet	38
A mellékletben található fájlok	39

Bevezetés

A projekt célja

Az *OOPS!* (**O**bject-**O**riented **P**HP **S**cript) fantázianévre keresztelt projekt célja, hogy olyan programozási eszköztárat biztosítson a fejlesztőknek, amellyel a legfrissebb szabványoknak (HTML5¹) megfelelő webes alkalmazásokat készíthetnek. Az említett szabvány használata garantálja, hogy az elkészült alkalmazás megállja helyét bármilyen platformon, legyen szó egy HD felbontású kijelzővel szerelt asztali számítógépről, egy táblagépről, vagy egy okostelefonról. Ez azért kiemelkedően fontos, mert napjainkban a mobil eszközök térhódítása következtében az ezen eszközökön történő weblap-megtekintések száma egy év alatt közel duplájára, nagyjából 12%-ra nőtt. [1]²



1. ábra Az asztali, illetve mobil eszközökről kezdeményezett böngészések trendje az utóbbi 1 évben

¹ <http://dev.w3.org/html5/>

² <http://gs.statcounter.com>

Az *OOPS!* library segítségével lehetőség van a legnépszerűbb szerver oldali programozási nyelv, a PHP egyes hiányosságainak kiküszöbölésére, egy objektumorientált környezetet biztosítva. Ezen felül egy hierarchikusan felépített osztályszerkezettel reprezentálva a HTML dokumentum elemeit lehetővé teszi webes alkalmazások HTML kód használata nélküli elkészítését, ezzel párhuzamosan garantálva a végeredményben keletkező weboldal szabványosságát a legújabb sztenderdek szerint.

Cél továbbá, hogy a programkönyvtár egyszerűen bővíthető legyen, akár a programozó saját igényei szerint is. Ebben nagy segítséget nyújtanak a megvalósított típusműveletek, valamint azok az absztrakt osztályok, interfészek, melyek kategorizálják alosztályaikat. Ezeket használva a fejlesztő által készített bővítmények egyszerűen beilleszthetők a rendszerbe úgy, hogy az megőrizze integritását.

Célközönség

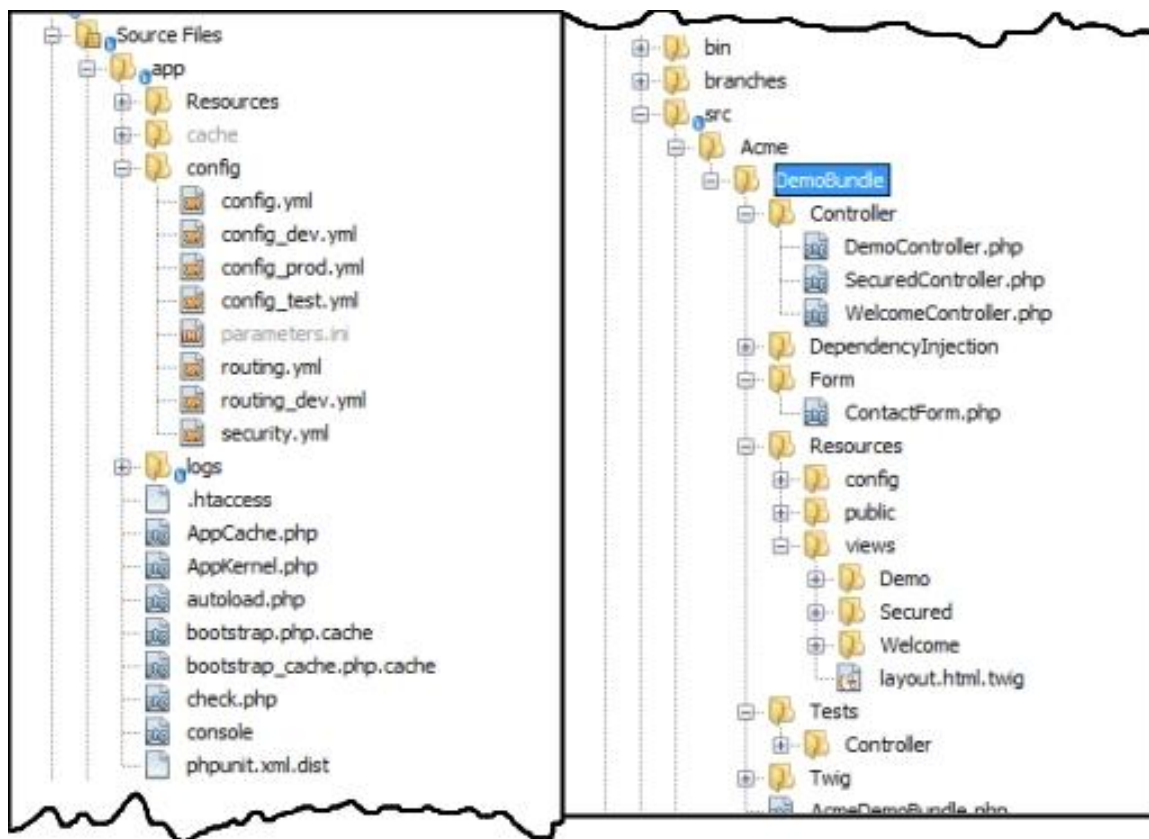
A kifejlesztett library egyértelműen a fejlesztőknek szól, az ő munkájukat hivatott megkönnyíteni. A beszédes osztálynevek és az egyszerű, logikus felépítés a könnyű tanulhatóságot segíti elő, így akár középhaladó, vagy akár kezdő programozók is könnyedén elsajátíthatják a programkönyvtár használatát. Ezzel együtt más programozási nyelvekben járatos programozók is könnyen megtanulhatják az eszköztár kezelését, mert azok a Java és a .NET nevezéktanára épülnek.

Hasonló rendszerek

Számos objektumorientált PHP programozást, űrlapépítést lehetővé tevő eszköz található a weben. Ezek közül a népszerűbbek (pl. Zend Framework [2]³, Symfony Framework [3]⁴) általában nagy, masszív keretrendszerek, melyek gyakran az MVC architektúrára épülnek, hosszadalmas installálási, beállítási folyamatot igényelnek. Ez a rendszer nyilvánvalóan nagy segítséget nyújthat egy nagy komplexitású web-alkalmazás fejlesztésében, de a fejlesztők egy tekintélyes részének nincsen szüksége ilyen összetett rendszerre ahhoz, hogy az általa tervezett alkalmazást megvalósítsa, ami azt eredményezi, hogy végül nem fogja ezeket a rendszereket használni. Ennek szemléltetésére tekintsük meg azt a mappaszerkezetet, amely egy friss telepítés után fogad minket a Symfony 2 keretrendszer esetében.

³ <http://framework.zend.com>

⁴ <http://symfony.com/what-is-symfony>



2. ábra Mappaszerkezet a Symfony 2 Framework telepítése után

Ezzel szemben, az *OOPS!* library egy olyan külső könyvtár, amelyet egyszerűen a projekthez csatolhatunk, ami nem jelent jelentős megkötéseket a programozó számára. Ez a programkönyvtár nem egy teljesen felépített mappaszerkezet ad a fejlesztőnek, melyben minden létrehozandó elemnek előre meghatározott helye van, hanem csak egyetlen mappát, mely a betölthető modulokat tartalmazza. Épp ezért kis- és középméretű projektekhez a legalkalmasabb, de bővíthetősége révén akár nagy szoftverek készítésére is használható.

Fontos szempont továbbá, hogy a HTML5 szabványnak való megfelelés garantált a library használatakor, szemben más egyéb rendszerekkel, melyek gyakran még az XHTML 1.1-et használják. A legújabb sztenderdek követése biztosítja az előrefele-kompatibilitást, és a multi platform működést. A szabványok betartása pedig fontos (de önmagában nem elegendő) lépést jelent az egyenlő esélyű hozzáférés támogatásához. A Web Akadálymentesítési Útmu-

tató 2.0 ajánlás [4]⁵ negyedik alapelve a robosztusságról szól, amelynek első irányelve a kompatibilitásra vonatkozik:

4.1 Irányelv: Kompatibilitás: Maximalizálja a kompatibilitást a jelenlegi és jövőbeli felhasználói programokkal, beleértve a kisegítő technológiákat is.

Az irányelv célja a W3C dokumentáció szerint [5]⁶:

„Jelen irányelv célja az, hogy támogassa a jelenlegi és jövőbeni felhasználói programok közötti kompatibilitást, különös tekintettel a segítő technológiákra (AT). Ezt a cél támogatja a 1)-es pont azzal, hogy nem engedi megsérteni (pl. rossz minőségű jelölés), illetve ki játszani a segítő technológiákat (pl. a gyakorlattól eltérő jelölés vagy kód használata), valamint a 2)-es pont, amely biztosítja azt, hogy a tartalomban a szabványoknak megfelelő és a segítő technikák által felismerhető, illetve kezelhető információ kerüljön feltüntetésre.”

Láthatjuk tehát, hogy a szabványosság az egyenlő esélyű hozzáféréshez szükséges feltételek között is jelen van.

Nyilvánvaló, hogy más, nagy fejlesztői bázissal rendelkező egyéb rendszerek által nyújtott szolgáltatások mennyiségével egyelőre nem versenyezhet ez az egyszemélyes projekt, de a cél, hogy a folyamatos fejlődés, fejlesztés, valamint az állandó magas minőség versenyképessé tegye ezt a programkönyvtárat a web-programozók körében.

⁵ <http://www.w3c.hu/forditasok/WCAG20/>

⁶ <http://www.w3c.hu/forditasok/UNDERSTANDING-WCAG20/complete.html>

A programkönyvtár felépítése

A céloknak megfelelően az *OOPS!* library jelenleg három modulra tagolódik: *Core*, *Form* és *Message*. A következőkben ezeknek a moduloknak a funkcióját, felépítését részletezzük.

A *Core* modul

A *Core* modul újradefiniálja az alapvető típusok (számok, karakterek, szövegek, tömbök, stb.) műveleteit objektumorientált módon, valamint további típusokat is megvalósít. Ez egy részről azért fontos, mert gyakran a PHP beépített függvényeinek neve kaotikus, zavaró lehet (pl. szövegkezelő függvények: `strstr()`, `str_replace()`, `substr()`), más részről pedig azért, mert ez adja meg a további modulok objektumorientált alapját. Ennek a modulnak a felépítése és nevezéktana a Java nyelv kapcsolódó részein alapszik.

A *Form* modul

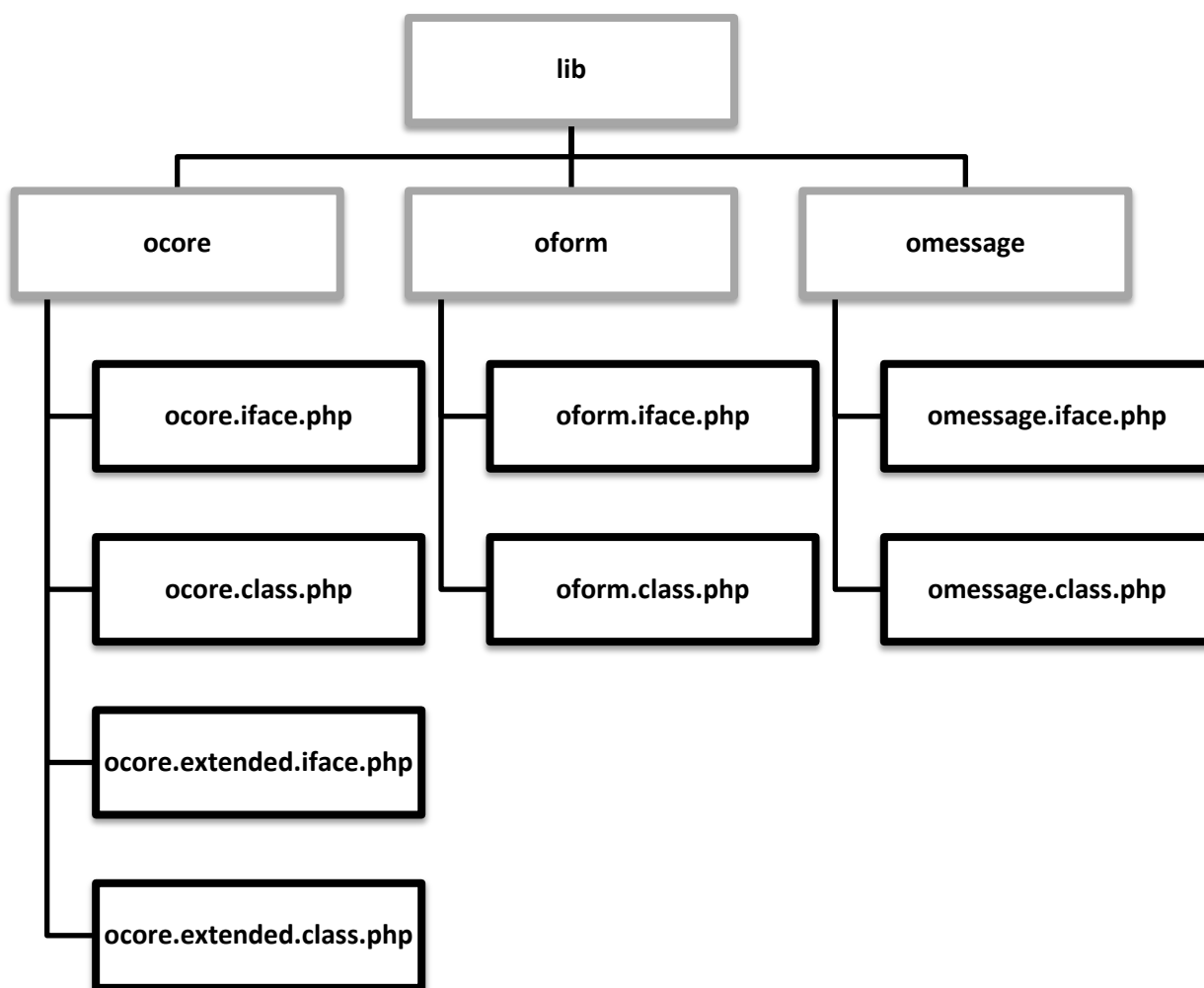
A *Form* modul definiál egy osztályszerkezetet, melynek elemei a HTML dokumentumok elemeit, illetve azok csoportjait reprezentálják. Ez lehetővé teszi a programozó számára, hogy objektumok segítségével építse fel a HTML oldalt, melyet a program futásának végén a programkönyvtár automatikusan legenerál és megjelenít. Ez a megoldás biztosítja végső soron a felépített dokumentum szabványoknak való megfelelést. A tervezésnél mintául a .NET keretrendszer űrlapjai szolgáltak, természetesen a HTML DOM felépítéséhez igazítva.

A *Message* modul

A *Message* modul a PHP által generált, a végfelhasználónak szánt üzeneteket kezeli, legyenek azok hibaüzenetek, kiegészítő információk, vagy egy művelet sikerét jelző szövegek. A modul használatának előfeltétele a *Form* modul használata, annak funkcionalitását bővíti ki. Azért fontos ez a komponens, mert működése garantált a *Form* modul azon funkciója mellett is, amely biztosítja, hogy a böngésző számára elküldött űrlap-információk frissítéskor ne legyenek újraküldve, ezáltal esetlegesen végrehajtva egy nemkívánatos módosítást.

Mappa- és fájlstruktúra

Az elkészített programkönyvtár mappaszerkezetét legjobban az egyszerű és hatékony szavakkal lehet jellemezni. A library-hez kapcsolódó minden forrásfájl a `lib` könyvtárban található. Ezen belül minden modulhoz tartozik egy mappa, ebben találhatóak a forrásfájlok. A pontos felépítést az alábbi ábra szemlélteti:



3. ábra A programkönyvtár felépítése

Osztályszerkezet

Ebben a részben a *Form* modul osztályszerkezetét fejtjük ki részletesen, mivel ez a modul a legkomplexebb felépítésű, és ez hordozza azt az egyedi megoldást, ami a programkönyvtár lényegét adja.

Control

Az osztály-hierarchia csúcsán a `Control` absztrakt osztály áll, mely definiálja az összes alosztály számára szükséges metódusokat, konstansokat. Ennek publikus része az alábbi `ioFormControl` interfész:

```
interface ioFormControl {
    const TABINDEX_AUTO = 0;
    const TABINDEX_SKIP = -1;
    public function __wakeup();
    public function addEvent();
    public function displayControl();
    public function getContainer();
    public function getId();
    public function getTabIndex();
    public static function isControl(&$value,
                                    $class='Control', $parameter=0);
    public function isDisplayed();
    public function isEnabled();
    public function setDisplayed($value);
    public function setEnabled($value);
    public function setTabIndex($value);
}
```

Három érdekesség, amit megfigyelhetünk az interfészen:

1. Szerepel benne egy statikus metódus, az `isControl`. Ez a *Core* modulban bevezetett sablonnak megfelelő módon definiál egy statikus típusellenőrző függvényt. Ennek a függvénynek egyedi tulajdonsága, hogy a paraméterként kapott `string` típusú változókat automatikusan konvertálja `Text` objektumokká, ezzel lehetővé téve a szövegek dokumentumhoz adásának egyszerűbb módját.

```
Document::addControl(new Text('Példa szöveg'));7
```

```
Document::addControl('Példa szöveg');
```

2. Szerepel benne a `__wakeup` „magic” metódus, mely biztosítja, hogy a szerializált változók kibontásakor azok `$container` attribútuma `NULL` értékre állítódik, ezzel lehetővé téve hogy azt újra csatoljuk egy `Container`-hez. Ez a funkció nagyon nagy szerepet játszik a `Message` objektumok oldalak közötti továbbításánál.
3. Nem szerepel benne se a konstruktor, sem pedig a `PARENT_BASE_CLASS` sem pedig az `ANCESTOR_BASE_CLASS` konstans. Ennek oka az, hogy a PHP nem teszi lehetővé interfészben létrehozott konstansok újradefiniálását, valamint metódusok más paraméterlistával történő túlterhelését.

Container

A másik, a szerkezetben kulcsszerepet betöltő absztrakt osztály a `Container`. Ennek interfésze (`iOFormContainer`) az alábbi módon van definiálva (az interfész a `Control` osztály interfészéből származik):

```
interface iOFormContainer extends iOFormControl {  
    public function addControl();  
    public function getControl($name);  
}
```

⁷ Az áthúzott kód azt a megoldást jelöli, mely helyett a következő sorban lévő egyszerűbb módszert használjuk

A `Container` minden olyan további osztály őssztálya, melyek további elemeket tartalmazhatnak. Ez az osztály kibővíti a `Control` funkcionalitását azzal a lehetőséggel, hogy az objektumhoz további gyerekelemek csatolhatóak. A megvalósítása ennek egy asszociatív tömb segítségével történik. Az azonosítóval rendelkező elemeket azonosítójuk alapján tároljuk el a tömbben, a további elemeket számozott kulcs alapján. Ez a módszer megkönnyíti a későbbiekben a már csatolt objektumok elérését. Ezen a ponton nagy előnyt jelent a PHP azon tulajdonsága, hogy függvényhívásnál az objektumokat mindig referencia szerint adja át. Ez lehetővé teszi azt, hogy ha egy változót már csatoltunk egy `Container`-hez, akkor továbbra is módosíthatjuk a változót, az értékváltozás meg fog jelenni a végeredményben is.

```
$urlap = new Form('urlap1', 'pelda.php');
$urlapElem = new TextBox('szoveg1', 'szoveg');
$urlap->addControl($urlapElem);
$urlap->getControl('szoveg1')->setPlaceholder('Ide írjon...');
$urlap->setPlaceholder('Ide írjon...');
```

Document

Kitüntetett helye van továbbá a `Document` osztálynak, mely az osztályhierarchia „alján” helyezkedik el. Eltér a többi osztálytól többek között abban, hogy egy ez egy „Singleton” osztály, vagyis csak egy példánya lehet, amely automatikusan létre is jön. Ennek az osztálynak a metódusait statikusan érhetjük el bárhonnan a kódunkból. Például, ha közvetlenül az oldalunk törzséhez kívánunk elemeket hozzáadni, azt az alábbi statikus hívással tehetjük meg. (paraméterként átadva azt az elemet (`Control` osztályból származó objektumot), melyet csatolni szeretnénk.

```
Document::addControl();
```

Ennek az osztálynak a megvalósítása is különleges. Maga a `Document` osztály absztrakt, és nem helyezkedik el a korábban említett osztályhierarchiában. Helyette a `DocumentInstance` osztály lett származtatva a `Panel` osztályból. A `Document` tárolja a `DocumentInstance` egy példányát, és a `__callStatic` magic metódus segítségével biztosítja, hogy `Document` osztályon végrehajtott statikus metódushívások továbbítódjanak a `DocumentInstance`-hoz.

Mivel a `Document` osztály absztrakt, ezért nincs neki konstruktora, mégis tartalmaz egy metódust, ami az inicializálást szolgálja, ez az `__init`. Ez a metódus tartalmazza a belső változók inicializálását (így a `DocumentInstance` létrehozását), valamint az egész oldalra vonatkozó beállításokat. Szintén ebben az `__init` eljárásban található az a kódrészlet, amely biztosítja, hogy külön hívás nélkül a teljes script lefutása után megjelenjen a legenerált HTML oldal. Erre a PHP által biztosított `register_shutdown_function` függvény ad lehetőséget.

```
register_shutdown_function(array('Document', 'displayControl'));
```

A kódrészletből láthatjuk, hogy a `register_shutdown_function` által meghívott eljárás a `Document` osztály `displayControl` metódusa. Mivel ez az eljárás nem létezik, ezért a `__callStatic` magic metódus automatikusan továbbítja a kérést a `DocumentInstance` felé, mely rekurzívan előállítja az oldalt.

A `register_shutdown_function` használatával kapcsolatban felmerülhet egy probléma: az oldal hibás működése esetén is meghívna a `displayControl` metódust, így egy részben előállított oldalt legenerálva. Ennek kiküszöbölésére nyilvántartjuk a `Document` osztályban, hogy történt-e hiba a script interpretálása során. Erre egy saját hibakezelő metódust hoztunk létre. Ezt a metódust a `set_error_handler` beépített php eljárás segítségével rögzítjük hibakezeléshez.

```
public static function errorHandler() {  
    self::$error = TRUE;  
    return FALSE;  
}
```

Ez az eljárás mindössze annyit tesz, hogy egy hibát jelző változót igazra állít. Ez alapján a változó alapján a `DocumentInstance::displayControl` metódus lényegében nem fog lefutni.

```
if (!Document::$error) :
```

A hamis visszatérési érték azért fontos, mert ez a PHP hibaértelmezőjének azt az üzenetet adja, hogy ettől a hibakezelő függvénytől függetlenül az alapértelmezett hibakezelőt is futtatni kell.

Bázisosztályok

Az osztályok további felépítése olyan, hogy a logikailag összetartozó, vagy hasonló viselkedésű osztályokat összefogja egy közös absztrakt ősosztállyal. Ennek a kódDuplikáció elkerülése mellett nagyon fontos szerepe van a szabványosság biztosításában is, mivel ezek alapján a közös ősosztályok alapján dönti el a program, hogy egy adott osztály (a `Container` osztály valamely leszármazottja) mely osztályok példányait tartalmazhatja. Ez a fajta ellenőrzés a másik irányba is működik, minden osztály nyilvántartja, hogy mely osztályok tartalmazhatják őt közvetlenül, illetve közvetetten. Az erre szolgáló konstansokat a `Control` osztály definiálja:

```
const PARENT_BASE_CLASS = 'Container';  
const ANCESTOR_BASE_CLASS = 'Container';
```

Ezeket a konstansokat az alosztályok szükség szerint felüldefiniálják. Példa erre a `FormControlBase` absztrakt osztály, ami az űrlapokon megjelenő elemek közös ősosztálya. Ennek az osztálynak a leszármazottjaival szemben mindig elvárás, hogy valamelyik ősük a HTML dokumentumban egy űrlap legyen. Ennek megfelelően definiálja felül az `ANCESTOR_BASE_CLASS` konstant a `FormControlBase` osztály.

```
abstract class FormControlBase extends Control implements  
    iOFormFormControlBase {  
    const ANCESTOR_BASE_CLASS = 'Form';
```

A dokumentum generálása

A dokumentum végső legenerálásában is fontos szerepet játszik ez a hierarchia. A `Control` osztály definiálja az absztrakt `displayControl` és `getAttributes` metódust. Mivel majdnem minden további osztály a `Control`-tól származik, ezért ezeknek mind meg kell valósítaniuk ezeket az eljárásokat.

```
abstract public function displayControl();  
  
protected function getAttributes();
```

A `displayControl` feladata, hogy előállítsa az adott elem kimeneti kódját. Ezen belül használjuk a `getAttributes` metódust, amely mindig úgy van megvalósítva, hogy az adott elem összes olyan attribútumának formázott listáját adja eredményül, melyek megjelenítésre

kerülnek az adott tag HTML kódjában. Ezzel a módszerrel elérhető, hogy ha nem szükséges, akkor nem jelenítjük meg azokat az attribútumokat, amelyek önmagukban módosítanák az elem tulajdonságait (pl. `disabled='disabled'` attribútum), illetve azokat, amelyek nem tartalmaznak értéket (pl. `placeholder=''`)

Ezt követően a `displayControl` rekurzív hívásával előállítható a kész HTML oldal. (Azok az osztályok, melyek nem a `Control`-tól származnak, azok nem HTML elemeket reprezentálnak azok vagy nem tartoznak ehhez a hierarchiához (`OFormException`) vagy csak attribútumokat reprezentálnak (`Event`), vagy közvetlenül csak a `Document`-hez csatolható elemek (`Stylesheet`, `Script`).

Bővíthetőség

A `getAttributes` metódus azért nagyon fontos, mert ez teszi nagyon egyszerűvé a rendszer bővítését új HTML elemekkel. Tekintve, hogy a hasonlóan viselkedő HTML tag-ek hasonló attribútumokkal rendelkeznek, ezért elegendő, ha a közös ősosztályukban megfelelően definiálva van a `getAttributes` metódus, arra a leszármazott osztály `displayControl` műveletében már egyszerűen tudunk hivatkozni.

Az egyszerű bővíthetőséget szolgálja a `Container` osztályban megvalósított `displayChildControls` metódus is, amely egy elemeket tartalmazó objektum összes gyerekelemét megjeleníti. Ez azért előnyös, mert így meghatározhatjuk, hogy egy `Container`-ból származtatott osztály kimenetében hol helyezkedjenek el a gyerekelemek.

Ezek után nézzünk egy példát az egyszerű bővíthetőségre. Tekintsük meg a HTML5-ben megjelent `<article>` tag megvalósítását a programkönyvtárban:

```

class Article extends ParagraphBase implements iOFormParagraphBase {
    public function __construct($id = '') {
        parent::__construct($id);
    }

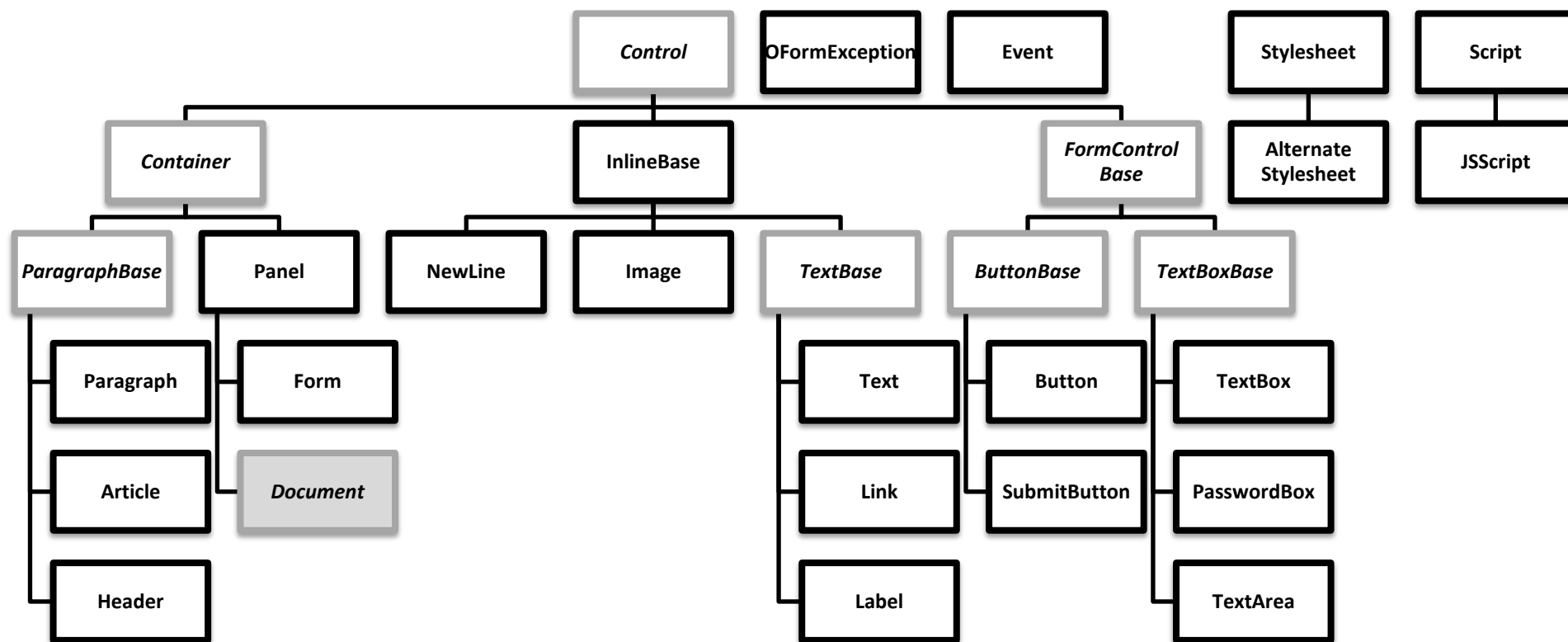
    public function displayControl() {
        echo '<article' . $this->getAttributes() . '>' . PHP_EOL;
        $this->displayChildControls();
        echo '</article>' . PHP_EOL;
    }
}

```

A rövid megoldásból is láthatjuk, hogy gyakorlatilag csak azt kellett manuálisan megadni, hogy melyik tag jelenjen meg a kimeneten, valamint, hogy hol helyezkedjenek el az attribútumai, illetve gyerekelemei. Látszik továbbá, hogy létrehoztunk egy interfészt is az adott osztályhoz. Az, hogy minden osztálynak van egy saját interfésze, amelyekben az osztály publikus metódusait definiáljuk, egy fontos konvenció a kódban, amit követnünk kell. Az interfészek között is fennáll ugyanaz a hierarchia, mint az osztályok között, így itt is csak azokat az eljárásokat kell definiálni, melyek egyik őosztályban sem voltak még definiálva.

Osztálydiagram

A következőkben megtekinthetjük a *Form* modul jelenleg megvalósított osztályainak osztálydiagramját. A diagramon vonalakkal vannak feltüntetve a származtatási kapcsolatok. A dőlt betűkkel írt, világosabb, szürke kerettel rendelkező osztályok absztraktak. Külön kiemelésre került a *Document* osztály, amely absztraktként van definiálva, de tárol egy példányt a *DocumentInstance* osztályból (ténylegesen ez az osztály van származtatva a *Panel* osztályból), ezzel egy „singleton” osztályt alkotva.



4. ábra A Form modul osztálydiagramja

A library használata

Egy példafeladat megoldása

A programkönyvtár működésének bemutatásához tekintsünk egy példafeladatot és annak megvalósítását a programkönyvtár segítségével. Habár a nemzetközi célközönségre való tekintettel a library forráskódja teljesen angol nyelvű, a példában bemutatott program során magyar változóneveket fogunk használni, ezzel is mutatva, hogy a használat során a programkönyvtár a programozóval szemben nem jelent ilyen téren megkötéseket. A példaprogram kipróbálható a CD mellékleten lévő fájlok segítségével. Részletes útmutató a kipróbáláshoz a *Melléklet* fejezetben.

A megoldandó feladat

Példánkban egy egyszerű vendégkönyvet szeretnénk készíteni oldalunkhoz. Szeretnénk továbbá, hogy ez az oldal több nyelven is elérhető legyen, valamint, hogy a gyengénlátó felhasználók is egyszerűen válthassanak számukra is könnyen olvasható nézetre.

A megoldásban külső fájlokat fogunk használni a nyelvi adatok, illetve a korábban elküldött üzenetek tárolására. Az utóbbiak egy CSV-szerű, tabulátorokkal tagolt állományból érhetőek el, ezek alapján fogjuk felépíteni az oldalunkat. Az akadálymentes nézetet alternatív stíluslap segítségével oldjuk meg, melynek neve mindig az aktuálisan kiválasztott nyelven jelenik meg a böngészőben a lehetőségek között.

A példa megvalósításában láthatjuk, hogyan épül fel egy HTML dokumentum, valamint több újradefiniált alaptípus-kezelő függvényt is láthatunk használatban.

Tekintsük át a megvalósítást lépésről lépésre:

Először a programunk elején betöltjük a használni kívánt modulok interfészeit, jelen esetben a *Message* modulét. Azért elegendő csak ezt az egy modult betölteni, mert a modulok megvalósítására jellemző, hogy automatikusan hivatkoznak minden olyan további modulra, amelyek szükségesek a működésükhöz. Példánkban ez azt jelenti, hogy a *Message* modul automatikusan csatolja a *Core* és *Form* modult is. Az osztályok megvalósítását tartalmazó fájlokat sem kell betöltenünk, mivel az interfészek a hozzájuk tartozó osztálydefiníciós fájlo-

kat is csatolják. Ez a megoldás felvet egy kérdést, miszerint nem okoz-e hibát, ha egy fájl többször kerül betöltésre, illetve nem okoz-e teljesítményvesztést. Ennek a problémának a megoldására a PHP `require_once` kulcsszavát használjuk, ami garantálja, hogy egy fájl, csak egyszer kerül csatolásra, valamint, hogy a program csak akkor fusson tovább, ha a csatolás sikeres volt. Ezek alapján a programunk a következő kóddal kezdődik:

```
<?php  
require_once("lib/omessage/omessage.iface.php");
```

A következő részben a tabulátorokkal tagolt nyelvi fájlt olvassuk be. Először kigyűjtjük az elérhető nyelvek listáját, amelyet a fájl első sora tartalmaz.. Ezt követően meghatározzuk, hogy van-e felhasználó által meghatározott nyelv (GET metódussal továbbítva), ha nincs, akkor egy alapértelmezett nyelvet használunk, mely az elérhető nyelvek közül az első. További tartalmából a kívánt nyelvhez tartozó szövegeket egy asszociatív tömbben tároljuk, melynek kulcsai mindig a sorok elején megadott értékek. A folyamat lépései:

1. A kód beszédesebbé tételéhez definiáljuk a `LANGUAGE_PATH` konstanst, ami a nyelvi fájl elérési útját tartalmazza.

```
define("LANGUAGE_PATH", "nyelvek.txt");
```

2. Létrehozunk egy `Panel` objektumot, ebben lesznek a nyelvváltáshoz szükséges linkek. Létrehozzuk a szövegek tárolására szánt üres tömböt.

```
$nyelvPanel = new Panel("nyelvek");
```

```
$szovegek = array();
```

3. `Foreach` ciklus segítségével végigmegyünk a nyelvi fájl sorain (az üres sorokat és a sorvégi sortörés karaktereket figyelmen kívül hagyjuk).

```
foreach (file(LANGUAGE_PATH, FILE_SKIP_EMPTY_LINES |  
FILE_IGNORE_NEW_LINES) as $sorszam => $sor) :
```

- a. A sorokat feldaraboljuk a tabulátorok mentén a `String::split()` függvény segítségével.

```
$sor = String::split($sor, "\t");
```

- b. Az első sornál eltároljuk az elérhető nyelveket egy változóban. Ezt követően eldöntjük, hogy mi az oldal nyelve, az alapján, hogy a `$_GET["nyelv"]` változó be van-e állítva.

```
if ($sorszam == 0) :  
    $nyelvek = $sor;  
    if (isset($_GET["nyelv"]) AND  
        Arrays::contains($nyelvek, $_GET["nyelv"])) :  
        $nyelv = $_GET["nyelv"];  
    else :  
        $nyelv = $nyelvek[0];  
endif;
```

- c. Minden elérhető nyelvhez létrehozunk egy új hivatkozást a korábban létrehozott Panel-hez.

```
foreach ($nyelvek as $nyelvkod) :  
    $nyelvPanel->addControl(new Link($nyelvkod,  
        "pelda.php?nyelv=$nyelvkod"));  
endforeach;
```

- d. A további soroknál kiolvassuk a kívánt nyelvhez tartozó szöveget a sorból, és eltároljuk az asszociatív \$szovegek tömbben a sor elején található kulcs alapján.

```
else :  
    Arrays::add($szovegek, Arrays::get($sor,  
        Arrays::indexOf($nyelvek, $nyelv) + 1), $sor[0]);  
endif;  
endforeach;
```

4. Végül az összeállított panelt, mely a nyelvek közötti váltásra szolgáló linkeket tartalmazza, hozzáadjuk a dokumentumhoz.

```
Document::addControl($nyelvPanel);
```

A program következő részében kezeljük az újonnan érkezett üzeneteket. Ehhez először létrehozunk egy konstanst, ami a bejegyzéseket tartalmazó fájl elérését tartalmazza, ugyanazon okból mint korábban a nyelvi fájl esetében.

```
define("FILE_PATH", "vendegkonyv.txt");
```

A következő lépésben egy elágazásban vizsgáljuk, hogy volt-e küldött üzenet (tartalmaz-e elemeket a `$_POST` tömb. Amennyiben igen, megnyitjuk a fájlunkat hozzáírásra, majd beleírjuk tabulátorokkal elválasztva az üzenet küldőjének nevét, a küldés idejét és magát az üzenetet. A sort egy új sor karakterrel zárjuk. (Az üzenetet előfeldolgozzuk, a `'<'` és `'>'` karaktereket lecseréljük HTML entitás kódjukra, ezzel védekezve a káros scriptek ellen, valamint a felhasználó által beütött új sorokat kicseréljük a HTML új sor karakterére.) A felhasználót üzenetben tájékoztatjuk a művelet sikerességéről. Ehhez a `Message` osztályt használjuk, melynek minden létrehozott példánya automatikusan tárolódik, és későbbiekben megjelenítésre elérhető.

```
if (Arrays::length($_POST) > 0) :  
    $file = fopen(FILE_PATH, "a");  
    String::replaceAll($_POST["uzenet"], "<", "&lt;");  
    String::replaceAll($_POST["uzenet"], ">", "&gt;");  
    String::replaceAll($_POST["uzenet"], "\n", NewLine::NEW_LINE);  
    if (fwrite($file, $_POST["nev"] . "\t" . date("Y. m. d. H:i:s") .  
        "\t" . $_POST["uzenet"] . PHP_EOL)) :  
        new Message($szovegek["siker"]);  
    else :  
        new Message($szovegek["sikertelen"]);  
    endif;  
    fclose($file);  
endif;
```

Ezt követően már nincs más dolgunk, mint beállítani és megjeleníteni a HTML dokumentum elemeit. Innentől már a korábban kigyűjtött nyelvi adatokkal dolgozunk. Elsőként a teljes dokumentumra vonatkozó beállításokat végezzük el. Ezek:

1. Az oldal nyelvének, címének beállítása,

```
Document::setLanguage($nyelv)->setTitle($szovegek["cim"]);
```

2. Stíluslapok hozzáadása az oldalhoz,

```
Document::addStylesheet("stilus.css",  
    new AlternateStylesheet("akadalymentes.css",  
        $szovegek["akadalymentes"]));
```

3. Az oldal fejlécében szereplő cím és az üzenetküldő űrlap hozzáadása,

```
Document::addControl($szovegek["alcim"],  
    $urlap = new Form("urlap", "pelda.php?nyelv=" . $nyelv));
```

4. Az űrlap feltöltése az egyes szintű címmel, a címkékkal és az űrlapelemekkel, valamint az esetlegesen kapott üzenetekkel,

```
$urlap->addControl(new Header(1, $szovegek["kuld"]),  
    new Label($szovegek["nevcimke"], "nev"), new NewLine(),  
    new TextBox("nev", "nev"), new NewLine(),  
    new Label($szovegek["uzenetcimke"], "uzenet"), new NewLine(),  
    new TextArea("uzenet", "uzenet"), new NewLine(),  
    new SubmitButton($szovegek["kuld"], Messages::getMessages()));
```

5. A kitöltendő űrlapelemekben található helyőrző szöveg beállítása, a kitöltés kötelezővé tétele.

```
$urlap->getControl("nev")->setPlaceholder($szovegek["nev"])  
->setRequired(TRUE);
```

```
$urlap->getControl("uzenet")->setPlaceholder($szovegek["uzenet"])  
->setRequired(TRUE);
```

A program végén nem maradt más dolgunk, mint az eddigi üzeneteket kiírni. Minden üzenetet önálló panelre (<div>-be) rakunk a jobb formázhatóság érdekében. Ehhez a korábbihoz hasonló módon végigmegyünk a bejegyzéseket tartalmazó fájlban, annak sorait tabulátorok mentén szabdadjuk. Minden üzenet panelje egyedi azonosítót kap, a fájlban lévő sorszáma alapján.

```

foreach (file(FILE_PATH, FILE_SKIP_EMPTY_LINES |
FILE_IGNORE_NEW_LINES) as $id => $post) :
    $post = String::split($post, "\t");
    Document::addControl(new Panel("post" . $id));
    Document::getControl("post" . $id)->
        addControl($post[0] . String::replaceAll($szovegek["irta"],
            "\n", "") . $post[1], new NewLine(), $post[2]);
endforeach;

```

Az itt tárgyalt megoldás teljes kódja a következő oldalon olvasható.

A példa megoldásának kódja

```
<?php
require_once("lib/omessage/omessage.iface.php");
define("LANGUAGE_PATH", "nyelvek.txt");
$nyelvPanel = new Panel("nyelvek");
$szovegek = array();
foreach (file(LANGUAGE_PATH, FILE_SKIP_EMPTY_LINES | FILE_IGNORE_NEW_LINES) as $sorszam => $sor) :
    $sor = String::split($sor, "\t");
    if ($sorszam == 0) :
        $nyelvek = $sor;
        if (isset($_GET["nyelv"]) AND Arrays::contains($nyelvek, $_GET["nyelv"])) :
            $nyelv = $_GET["nyelv"];
        else :
            $nyelv = $nyelvek[0];
        endif;
        foreach ($nyelvek as $nyelvkod) :
            $nyelvPanel->addControl(new Link($nyelvkod, "pelda.php?nyelv=$nyelvkod"));
        endforeach;
    else :
        Arrays::add($szovegek, Arrays::get($sor, Arrays::indexOf($nyelvek, $nyelv) + 1), $sor[0]);
    endif;
endforeach;
Document::addControl($nyelvPanel);
define("FILE_PATH", "vendegkonyv.txt");
if (Arrays::length($_POST) > 0) :
    $file = fopen(FILE_PATH, "a");
    String::replaceAll($_POST["uzenet"], "<", "&lt;");
    String::replaceAll($_POST["uzenet"], ">", "&gt;");
    String::replaceAll($_POST["uzenet"], "\n", NewLine::NEW_LINE);
    if (fwrite($file, $_POST["nev"] . "\t" . date("Y. m. d. H:i:s") . "\t" .
        $_POST["uzenet"] . PHP_EOL)) :
```

```

        new Message($szovegek["siker"]);
    else :
        new Message($szovegek["sikertelen"]);
    endif;
    fclose($file);
endif;
Document::setLanguage($nyelv)->setTitle($szovegek["cim"]);
Document::addStylesheet("stilus.css",
new AlternateStylesheet("akadalymentes.css", $szovegek["akadalymentes"]));
Document::addControl($szovegek["alcim"], $urlap = new Form("urlap", "pelda.php?nyelv=" . $nyelv));
$urlap->
    addControl(new Header(1, $szovegek["kuld"]), new Label($szovegek["nevcimke"], "nev"),
    new NewLine(), new TextBox("nev", "nev"), new NewLine(),
        new Label($szovegek["uzenetcimke"], "uzenet"), new NewLine(),
        new TextArea("uzenet", "uzenet"), new NewLine(),
        new SubmitButton($szovegek["kuld"]),
        Messages::getMessages());
$urlap->getControl("nev")->setPlaceholder($szovegek["nev"])->setRequired(TRUE);
$urlap->getControl("uzenet")->setPlaceholder($szovegek["uzenet"])->setRequired(TRUE);
foreach (file(FILE_PATH, FILE_SKIP_EMPTY_LINES | FILE_IGNORE_NEW_LINES) as $id => $post) :
    $post = String::split($post, "\t");
    Document::addControl(new Panel("post" . $id));
    Document::getControl("post" . $id)->
        addControl($post[0] . String::replaceAll($szovegek["irta"], "\n", "") . $post[1],
        new NewLine(), $post[2]);
endforeach;
?>8

```

⁸ A kódban a foreach-endforeach / if-endif, valamint az AND / OR szintaxist használjuk, melyre a PHP nyelv ad lehetőséget. Ettől függetlenül használható a C-szintaxisban megszokott '{' / '}', illetve '&&' / '||' szintaxis.

Képernyőkép a megoldásról

Az alább látható képernyőképen az oldal kínai (egyszerűsített) nyelven történő megjelenítését láthatjuk alapértelmezett, illetve akadálymentesített stílus esetén. A stíluslapok külön .css fájlokban találhatóak, ezzel érvényesítve a tartalom és megjelenés különválasztását célzó programozási elvet.



5. ábra Képernyőkép a kész példaalkalmazásról alapértelmezett-, illetve akadálymentes nézetben.

A kimeneti HTML kód

A programkönyvtár által generált HTML kód megfelel az HTML5 szabványnak⁹ (illetve annak jelenleg publikált változatának, tekintve hogy ez a szabvány még nincs kész, folyamatosan változik). Ezek mellett a kód teljesíti az XML szabvány által állított követelményeket is (melyek sokkal szigorúbbak a HTML5 követelményeinél), ezáltal a kód XHTML5 validnak tekinthető.

A fent megvalósított példaprogram által generált HTML kód a következő:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="HU">
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1" />
<title>Vendégkönyv - Példaprogram</title>
<link rel="stylesheet" type="text/css" href="stilus.css" media="all" />
<link rel="alternate stylesheet" type="text/css" title="Akadálymentesített
változat" href="akadalymentes.css" media="all" />
</head>
<body>
<div id="nyelvek">
<a href="pelda.php?nyelv=HU">HU</a><a href="pelda.php?nyelv=EN">EN</a><a
href="pelda.php?nyelv=RU">RU</a><a href="pelda.php?nyelv=ZH">ZH</a></div>
<span>Vendégkönyv - írjon nekünk!</span>
<form method="post" action="pelda.php?nyelv=HU" id="urlap">
<h1>
<span>Üzenet küldése</span>
</h1>
```

⁹ A <http://validator.nu> oldal ellenőrzőjét használtuk a teszteléshez

```

<label for="nev">Neve:</label><br />
<input type="text" id="nev" name="nev" placeholder="Az Ön neve" />
<br />
<label for="uzenet">Üzenete:</label><br />
<textarea id="uzenet" name="uzenet" placeholder="Az Ön üzenete"></textarea>
<br />
<input type="submit" value="Üzenet küldése" />
<div id="MessageBox">
</div>
</form>
<div id="post0">
<span>Visnovitz Márton írta ekkor: 2012. 11. 21. 04:05:58</span>
<br />
<span>Üzenet a vendégkönyvbe.</span>
</div>
<div id="post1">
<span>Vendég írta ekkor: 2012. 11. 21. 04:06:47</span>
<br />
<span>Újabb üzenet.
<br />Üdvözlettel:
<br />Vendég</span>
</div>
</body>
</html>

```

A kimeneti HTML dokumentum kódját a validátorba másolva ellenőrizhetjük, az eredmény tényleg megfelel a jelenlegi legfrissebb szabványoknak:

1. **Info:** The Content-Type was text/html. Using the HTML parser.
2. **Info:** Using the schema for HTML5 + SVG 1.1 + MathML 3.0 + RDFa Lite 1.1.

The document validates according to the specified schema(s) and to additional constraints checked by the validator.

6. ábra Képernyőkép a példaalkalmazás validálásának eredményéről

Egyedi megoldások a programban

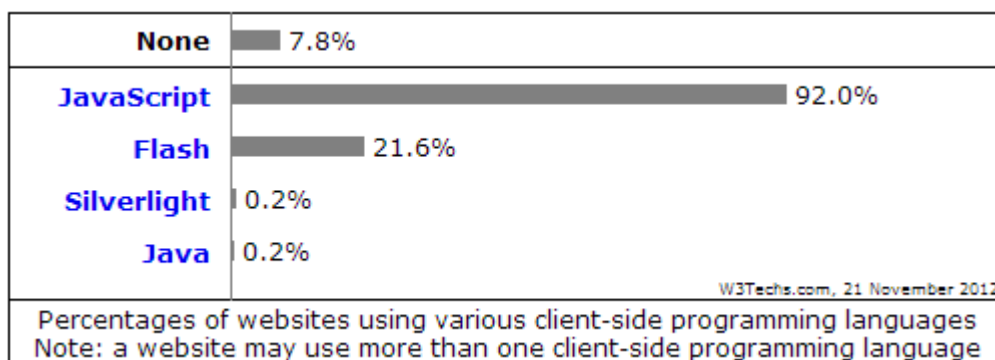
A program megvalósításakor a programozó munkájának egyszerűsítésére néhány kényelmi megoldás is alkalmazható.

Automatikus konverziók

A programkönyvtár néhány esetben lehetővé teszi, hogy objektumok helyett szöveg típusú adattal lehessen elemeket csatolni egy másik objektumhoz. Ennek egyik példája a már korábban is említett szöveg – Text objektum konverzió. Ez azért nagyon előnyös, mert a Text, vagyis az egyszerű szöveg a leggyakrabban előforduló elem egy HTML dokumentumban, ezért fontos, hogy minél egyszerűbb legyen ennek az elemnek a kezelése.

Hasonló megoldást alkalmaztunk a Document osztály addStylesheet és addScript metódusa esetén. Stíluslap hozzáadásánál, ha a paraméter csak egy egyszerű szöveg, akkor azt automatikusan Stylesheet objektummá konvertálja a program. Ezzel a megoldással elegendő csak akkor kézzel példányosítani a Stylesheet osztályt, ha médiát is szeretnénk hozzáadni, illetve, ha alternatív stíluslapot szeretnénk csatolni. (AlternateStylesheet)

Szintén ugyanez a megoldás külső script fájlok csatolásakor. A szöveg típusú paraméter automatikusan konvertálódik JSScript típusú objektummá, tekintve, hogy leggyakrabban JavaScript-et alkalmaznak a fejlesztők az oldalon. (Az oldalak kb. 92%-a használ JavaScriptet [6]¹⁰)



7. ábra Statisztika az egyes kliens oldali scriptnyelvek gyakoriságáról a weben

¹⁰ http://w3techs.com/technologies/overview/client_side_language/all

Metódusláncolás

Az *OOPS!* library tartalmaz egy olyan megoldást, ami lehetővé teszi bizonyos metódushívások láncolását. Ez azt jelenti, hogy egy objektumnak több metódusát is meghívhatjuk egymás után anélkül, hogy újból hivatkoznunk kellene az objektumra. Pl.:

```
Document::setLanguage($nyelv)->setTitle($szovegek["cim"]);
```

Ez alól természetesen kivételt jelentenek például a „getter” metódusok, hiszen ezek számunkra fontos információkkal térnek vissza.

Fejlesztési lehetőségek

Az elkészített programkönyvtár számos területen továbbfejleszthető. A HTML dokumentum elemeit reprezentáló osztályok a teljesség igénye nélkül lettek megvalósítva a projekt demonstrációjához. Számos elemnek még nincsen megfelelő reprezentációja programkönyvtárban. Ezen osztályok létrehozása azonban egyszerűen kivitelezhető a mögöttük álló rendszer segítségével.

A *Core* modul is tovább bővíthető további, már példányosítható típusok megvalósításával. A jövőben megvalósítható típusok listája már szerepel a forráskódban. Ezek az osztályok a teljesség igénye nélkül:

- *Date*: egy dátum osztály, mely tartalmazza a dátumokkal kapcsolatos műveleteket, dinamikusan kezeli az időzónákat, valamint dátumformátumokat.
- *Color*: egy színeket reprezentáló osztály, mely kezeli a különböző színkeverési eljárásokat, színformátumokat, átlátszóságot.
- *ArrayList*: a Java nyelvből ismert osztály, mely adott típusú elemek listáját tartalmazza.
- *Set*: a programozásban gyakran használt halmaz osztály, és a kapcsolódó műveletek.

Ezekon kívül a távlati tervek között szerepel az weboldalakon gyakran használt JavaScript programozásának lehetőségének megvalósítása a PHP keretein belül. Ez azért lenne előnyös, mert így egységesebbé válhatna a kód, nem kellene külön programozni a JavaScript kódot. Ez kezdő programozók számára is előnyös lehet, mert így csak egy nyelvet szükséges megtanulniuk.

Nyilvánvaló továbbá, hogy az elkészített kód nem hibamentes, nem volt alkalom a több mint száz metódus teljes körű tesztelésére. A használat során előkerülő hibák javítása is fontos feladat az *OOPS!* fejlesztése szempontjából. Ennek segítségét szolgálja, hogy a projekt helyet kapott a SourceForge¹¹ oldalon, ahol egy hibabejelentő oldal is csatolva lett a projekthez, hogy az esetleges felhasználók jelezni tudják a használat közben felmerült hibákat.

¹¹ <http://sourceforge.net>

A programkönyvtár terjesztése

Az *OOPS!* library nyilvános programozói eszköznek lett tervezve, ennek megfelelően gondoskodni kell, hogy béta és stabil változatai eljussanak a célközönséghez. Erre a célra lett létrehozva a SourceForge oldalon egy projekt a programkönyvtár részére. Ez lehetővé teszi *OOPS!* terjesztését, valamint, hogy eljusson azokhoz a potenciális fejlesztőkhöz, akik további modulokkal kívánják bővíteni a programkönyvtárat.

A SourceForge projekt számos módon segíti a program fejlődését. A szabad forráskódú szoftverek terjesztésére létrehozott oldal rengeteg hasznos funkcióval rendelkezik:

1. A kód számára biztosít egy online Git repository-t, ezzel biztosítva a folyamatosan bővülő kód verziókövetését.
2. Helyet ad a projekt közzétételre szánt változatainak, hogy azokat potenciális felhasználók letölthessék, használhassák.
3. Tartalmaz egy wiki oldalt, melyen kialakítható a programkönyvtárhoz a teljes online dokumentáció.
4. Biztosít egy hibabejelentő rendszert, melyen keresztül a felhasználók jelezhetik a használat közben felmerülő esetleges hibákat.
5. Létre lehet hozni rajta fórumokat, ahol a felhasználók elmondhatják véleményüket a projektről, ötleteket adhatnak a további fejlesztésekkel kapcsolatban, jelezhetik felmerülő igényeiket.

A programkönyvtárhoz tartozó SourceForge projekt megtalálható az alábbi webcímen:

<http://sourceforge.net/p/oopsphp>

Összefoglalás

Dolgozatomban bemutattam az általam kidolgozott és elkészített PHP programkönyvtárat, annak felépítését, kiemelve néhány fontosabb osztály megvalósítását. Kitértem más hasonló rendszerekkel való összehasonlítására, azokkal szembeni előnyeire, hátrányaira. Egy példaprogramon keresztül részletesen bemutattam az eredmény gyakorlatbeli alkalmazását, valamint bemutattam a library továbbfejlesztésének irányvonalát.

Eredmény

Munkám eredményeképp létrejött egy olyan, folyamatosan fejlődő, fejlesztőknek szánt web-programozói eszközkészlet, melynek segítségével szabad forráskódú szoftvereket használva egyszerűen, logikus, jól érthető kód segítségével hozhatóak létre kis-, közép- és nagyméretű web-alkalmazások, melyekben garantált a legújabb szabványoknak való megfelelés, ezáltal a kompatibilitás a legújabb eszközökkel, szoftverekkel.

A projekt keretében kibővítettem a PHP által nyújtott lehetőségeket azzal, hogy objektum-orientált környezetet biztosítok egy olyan nyelvben, amely ugyan támogatja az objektum-orientált programozást, de alapvetően nem erre az alapelvre épül. Ez a modul önállóan is használható azok számára, akik nem kívánják a programkönyvtár dokumentum-építő funkciót használni, csupán egy logikusabban felépített rendszerű, szabad forráskódú nyelv segítségével szeretnék webes alkalmazásokat készíteni.

Rendszerkövetelmények

A programkönyvtár használatához web-szerveren¹² kívül a PHP legalább 5.3-as változata szükséges. Ez letölthető a PHP hivatalos weboldaláról.¹³

¹² Pl.: Apache Webszerver : <http://httpd.apache.org>

¹³ <http://www.php.net>

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani témavezetőmnek, Abonyi-Tóth Andornak, aki rengeteg segítséget nyújtott a dolgozat elkészítésében, és aki szakmai tanácsaival segítette a projekt készítését. Köszönöm továbbá neki a webes szabványokkal, akadálymentesítéssel kapcsolatos óráin átnyújtott tudást, melyek többek között inspirálták munkám témaválasztását.

Köszönöm továbbá a projekt készítésében nyújtott segítségét Híves Áronnak, aki észrevételeivel, ötleteivel nagy mértékben segítette a programkönyvtár fejlesztését.

Szeretnék továbbá köszönetet mondani Varga Gábornak, aki elsőként vezetett be a PHP, azon belül is az objektum-orientált PHP programozásba. Az általa átadott ismeretek nélkül nem jöhetett volna létre ez a projekt.

Irodalomjegyzék

[1] *Mobile vs. Desktop from Oct 2011 to Oct 2012* | StatCounter Global Stats. (2012. november 21). Letöltés dátuma: 2012. november 21, forrás: <http://gs.statcounter.com/>

[2] *Zend Framework*. (dátum nélk.). Letöltés dátuma: 2012. november 21, forrás: <http://framework.zend.com/>

[3] *What is Symfony - Symfony*. (dátum nélk.). Letöltés dátuma: 2012. november 21, forrás: <http://symfony.com/what-is-symfony>

[4] *Web Content Accessibility Guidelines (WCAG) 2.0*. (2008. december 11). Letöltés dátuma: 2012. november 21, forrás: <http://www.w3c.hu/forditasok/WCAG20/>

[5] *A WCAG 2.0 Értelmezése*. (2008. december 11). Letöltés dátuma: 2012. november 21, forrás: <http://www.w3c.hu/forditasok/UNDERSTANDING-WCAG20/complete.html>

[6] *Usage Statistics of Client-side Programming Languages for Websites, November 2012*. (2012. november 21). Letöltés dátuma: 2012. november 21, forrás: http://w3techs.com/technologies/overview/client_side_language/all

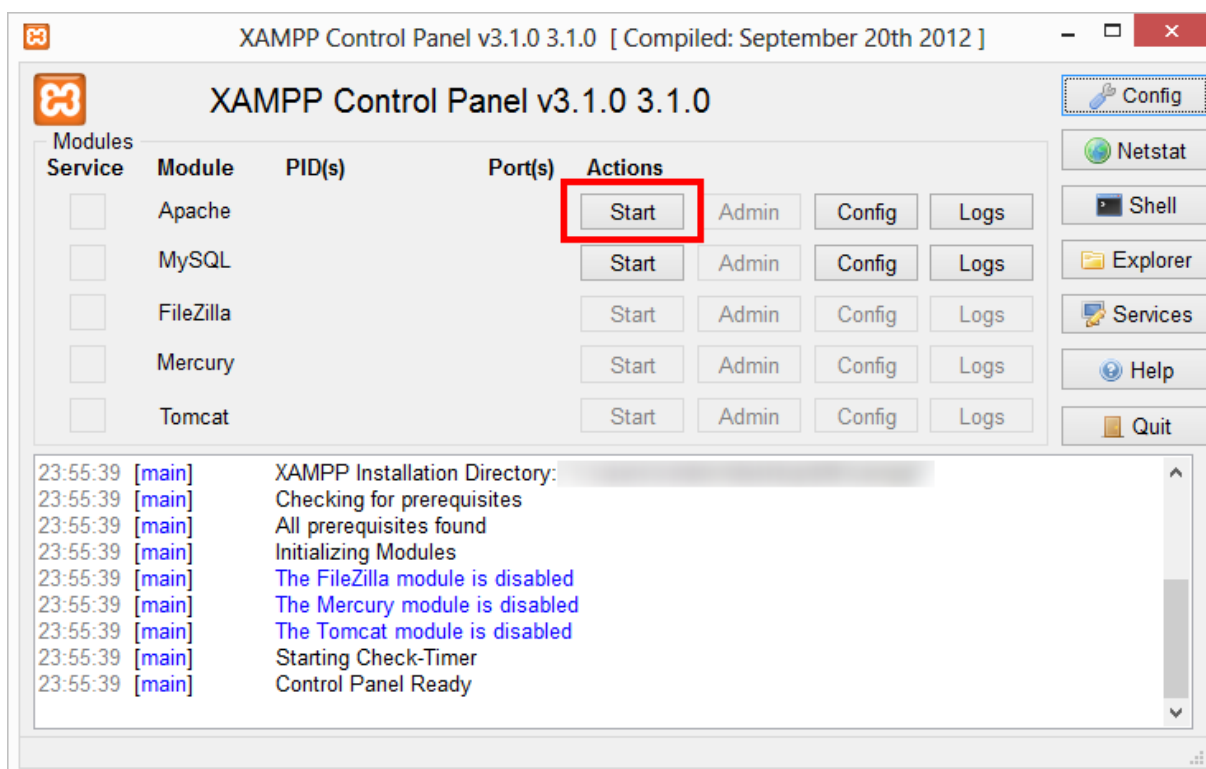
Ábrajegyzék

1. ábra Az asztali, illetve mobil eszközökről kezdeményezett böngészések trendje az utóbbi 1 évben	5
2. ábra Mappaszerkezet a Symfony 2 Framework telepítése után	7
3. ábra A programkönyvtár felépítése	10
4. ábra A Form modul osztálydiagramja	18
5. ábra Képernyőkép a kész példaalkalmazásról alapértelmezett-, illetve akadálymentes nézetben.	27
6. ábra Képernyőkép a példaalkalmazás validálásának eredményéről.....	29
7. ábra Statisztika az egyes kliens oldali scriptnyelvek gyakoriságáról a weben.....	30
8. ábra Az XAMPP grafikus kezelőfelülete	38

Melléklet

A dokumentumhoz mellékelte CD lemezen megtalálható a megvalósított programkönyvtár kódja, illetve a dolgozatban tárgyalt példaprogram megvalósítva, csatolva a működéshez szükséges fájlokkal, külső stíluslapokkal.

A CD lemez tartalmazza továbbá az XAMPP nevű ingyenes web-szerver csomagot¹⁴, mely lehetővé teszi, hogy saját web-szerver telepítése nélkül is futtassunk PHP scriptet Windows operációs rendszer alatt. Használatához másoljuk át a lemez teljes tartalmát a helyi géptetszőleges mappájába, majd futtassuk a `xampp/setup_xampp.bat` fájlt, kövessük a képernyőn leírtakat, majd indítsuk el a `xampp/xampp-control.exe`¹⁵ állományt. A megjelenő kezelőfelületen kattintsunk a Start gombra az Apache web-szerver sorában. Amennyiben a rendszer jogosultságot kér a web-szerver számára az internet eléréséhez, ezt nyugodtan megtagadhatjuk anélkül, hogy a rendszer működését befolyásolná.



8. ábra Az XAMPP grafikus kezelőfelülete

¹⁴ Forrás: <http://www.apachefriends.org/en/xampp.html>

¹⁵ A célmappához képest relatív útvonal

Fontos, hogy a program működéséhez a 80-as és 443-as portnak szabadnak kell lennie. Amennyiben ezek a portok már foglaltak, valószínű, hogy a számítógépen már fut egy web-szerver alkalmazás. Ez esetben használja saját web-szerverét a teszteléshez.¹⁶ Ehhez csupán másolja a CD-n található fájlokat (a xampp mappa kivételével) a számítógépen beállított web-szerver DocumentRoot (linux rendszerek esetében általában a /var/www) mappájába, és a szerveren beállított elérési úton keresztül (általában <http://localhost>) nyissa meg a böngészőprogramban.

Miután a program jelzi, hogy a szerver elindítása sikeres volt, indítsunk el egy böngészőprogramot,¹⁷ majd a címsorba gépeljük be az alábbi elérési utat:

<http://localhost/>

A mellékletben található fájlok

- [DIR] **lib**
 - [DIR] **ocore**
 - **ocore.extended.class.php**
 - **ocore.extended.iface.php**
 - **ocore.class.php**
 - **ocore.iface.php**
 - [DIR] **oform**
 - **oform.class.php**
 - **oform.iface.php**
 - [DIR] **omessage**
 - **omessage.class.php**
 - **omessage.iface.php**
- [DIR] **XAMPP**¹⁸
- **akadalymentes.css**
- **nyelvek.txt**
- **pelda.php**
- **pelda.phps**
- **stilus.css**
- **vendegkonyv.txt**

¹⁶ A script futtatásához a PHP legalább 5.3-as verziója szükséges

¹⁷ HTML5 szabványt támogató böngészőprogram szükséges (ajánlott a Mozilla Firefox vagy a Google Chrome böngésző legfrissebb változata)

¹⁸ Ennek a mappának a további tartalmát nem szemléltetjük