PROGRAMOZÁSI ELVEK

MONOLIT

OBJEKTUM ORIENTÁLTSÁG
MÉG NAGYOBB MONOLIT

PROBLÉMÁK AZ ÖRÖKLŐDÉSSEL

```php
abstract class Hero {
    public function attack() {
        echo "Smash the enemy with a sword!";
    }

    public function defend() { ... }
}
```

```php
abstract class Hero {
    public function attack() {
        echo "Smash the enemy with a sword!";
    }
    public function defend() { ... }
}

class Warrior extends Hero {}
class Viking extends Hero {}

$erik = new Viking();
$erik->attack(); // => "Smash the enemy with a sword!";
```

```php
class Mage extends Hero {
    public function attack() {
        echo "Cast a spell on the enemy!";
    }
}

$tim = new Mage();
$tim->attack(); // => "Cast a spell on the enemy!"
```

```php
class Mage extends Hero {
    public function attack() { ... }

    public function shave() {
        echo "Yo, I'm shaving!";
    }
}

$tim = new Mage();
$tim->shave(); // => "Yo, I'm shaving!"
```

```php
class Mage extends Hero {
    public function attack() {
        echo "Cast a spell on the enemy!";
    }
    public function shave() {
        echo "Yo, I'm shaving!";
    }
}


class Witch extends Mage {
    public function shave() {
        throw new Exception("Invalid operation!"); // :(
    }
}


class Witch extends Hero {
    public function attack() {
        echo "Cast a spell on the enemy!"; // :(
    }
}
```

KOMPOZÍCIÓ AZ ÖRÖKLŐDÉS FELETT

```php
interface AttackAbility {
    public function attack();
}

class SwordAttack implements AttackAbility {
    public function attack() {
        echo "Smash the enemy with a sword!";
    }
}

class SpellAttack implements AttackAbility {
    public function attack() {
        echo "Cast a spell on the enemy!";
    }
}
```

```php
abstract class Hero {
    protected $attackAbility;

    function __construct(AttackAbility $attackAbility = null) {
        $this->attackAbility = $attackAbility ?
            $attackAbility : new SwordAttack();
    }

    public function attack() {
        $this->attackAbility->attack();
    }
}
```

```php
class Warrior extends Hero {}

class Mage extends Hero {
    function __construct() {
        parent::__construct(new SpellAttack());
    }
}

class Witch extends Hero {
    function __construct() {
        parent::__construct(new SpellAttack());
    }
}

$tim = new Mage();
$tim->attack(); // => "Cast a spell on the enemy!"
```

```php
abstract class Hero {
    private $attackAbility;

    function __construct(AttackAbility $attackAbility = null) {
        $this->setAttackAbility($attackAbility);
    }

    public function attack() {
        $this->attackAbility->attack();
    }

    public function setAttackAbility(AttackAbility $attackAbility)
        $this->attackAbility = $attackAbility ?
            $attackAbility : new SwordAttack();
    }
}
```

MÓDOSÍTÁS VALÓS IDŐBEN!

```
$tim = new Mage();
$tim->attack(); // => "Cast a spell on the enemy!"

$tim->setAttackAbility(new SwordAttack());
$time->attack(); // => "Smash the enemy with a sword!";
```

```php
$erik = new Viking();

$erik->setAbility(new AttackAbility(array(
    'damageMultiplier' => 2,
    'twoHanded' => true
)));

$erik->setAbility(new DefendAbility(array(
    'absorbRate' => 5,
    'canUseShield' => false
)));

$erik->setItem(new SmallShield());
// exception thrown: can't use shield!
```

SOLID

SOLID

SINGLE RESPONSIBILITY PRINCIPLE

```php
class Hero {
    private $name;
    public function __construct($name) {
        $this->name = $name;
    }
    public function displayName() {
        echo "My name is " . $this->getName();
    }
    private function getName() {
        return $this->name . " the Hero";
    }
}

$erik = new Hero('Erik');
$erik->displayName(); // => STDOUT: "My name is Erik the Hero"
```

```php
class HeroDisplayer {
    public function show(Hero $hero) {
        echo "My name is " . $hero->getName();
    }
}

class Hero {
    private $name;
    public function __construct($name) { ... }
    public function getName() {
        return $this->name . " the Hero";
    }
}

$erik = new Hero('Erik');
$displayer = new HeroDisplayer();
$displayer->show($erik); // => STDOUT: "My name is Erik the Hero"
```

```php
class Book {
    private $pages = array();

    public function getTitle() { ... }

    public function turnPage() {
        // pointer to next page
    }

    public function getCurrentPage() {
        // returns the content of the current page
    }

    public function getLocation() {
        // returns the position in the library from db
        // ie. shelf number & room number
    }
}
```

```php
class Book {
    public function getTitle() { ... }
    public function turnPage() { ... }
    public function getCurrentPage() { ... }
}

class BookLocator {
    private $library;

    public function __construct(Library $library) {
        $this->library = $library;
    }

    public function locate(Book $book) {
        return $this->library->findPositionBy($book->getTitle());
    }
}
```

```
interface IEmail {
        public function setSender($sender);
        public function setReceiver($receiver);
        public function setContent($content);
}

class Email implements IEmail {
        public function setSender($sender) { ... }
        public function setReceiver($receiver ) { ... }
        public function setContent($content) { ... }
}
```

```php
interface IEmail {
        public function setSender($sender);
        public function setReceiver($receiver);
        public function setContent(IContent $content);
}

interface IContent {
        public function asString();
}

class Email implements IEmail {
        public function setSender($sender) { ... }
        public function setReceiver($receiver) { ... }
        public function setContent(IContent $content) {
            $textContent = $content->asString();
            // ...
        }
}
```

AVOIDING PRIMITIVE OBSESSION

```
$board = new Board();
$board->addCell(1,2);
$board->addCell(2,3);
$board->getNeighbours(2,3); // => [[1,2], [1,3], [1,4],
                            //      [2,1], [2,4], ...]
$board->moveCell(2,3, 2,4);


$board = new Board();
$board->addCell(1,2,2);
$board->addCell(2,3,3);
$board->moveCell(2,3,3, 2,4,3);
```

```php
$c1 = new Coordinate(1,2);
$c2 = new Coordinate(2,3);
$board->addCellTo($c1);
$board->addCellTo($c2);
$board->moveCell($c2, new Coordinate(2,4));


$board->getNeighbours($c2);
// => [Coordinate[1,2], Coordinate[1,3],
//     Coordinate[1,4], ...]


$c1 = new Coordinate(2,3,3);
$board->addCell(new Coordinate(1,2,2));
$board->addCell($c1);
$board->moveCell($c1, new Coordinate(2,4,3));
```

```php
class Cell {
    private $coordinate;

    public function __construct(Coordinate $coordinate)
        $this->coordinate = $coordinate;
    }

    public static function createAt($x, $y) {
        return new Cell(new Coordinate($x, $y));
    }
}

$board = new Board();
$cell = Cell.createAt(2,3);
$board->addCell($cell);
$board->moveCell($cell, new Coordinate(2,4));

$cell->getNeighbours();
// => [Cell[1,2], Cell[1,3], Cell[1,4], ... ]
```

SOLID

OPEN/CLOSED PRINCIPLE

```php
abstract class Shape {
    public $type;
    public $size;
}

class Rectangle extends Shape {
    public function __construct($size) {
        $this->type = 1;
        $this->size = $size;
    }
}

class Circle extends Shape {
    public function __construct($size) {
        $this->type = 2;
        $this->size = $size;
    }
}
```

```php
class Photoshop {

    public function drawShape(Shape $shape) {
        if ($shape->type === 1) {
            $this->drawRectangle($shape);
        } else if ($shape->type === 2) {
            $this->drawCircle($shape);
        }
    }

    private function drawRectangle($shape) { ... }
    private function drawCircle($shape) { ... }

}
```

```php
abstract class Shape {
    public function draw() {}
}

class Rectangle extends Shape {
    public function __construct($size) {
        $this->size = $size;
    }

    public function draw() { ... }
}


class Photoshop {
    public function drawShape(Shape $shape) {
        $shape->draw();
    }
}
```

SOLID

LISKOV'S SUBSTITUTION PRINCIPLE

Ha S altípusa T-nek, akkor minden olyan helyen ahol T-t felhasználjuk S-t is minden gond nélkül behelyettesíthetjük anélkül, hogy a programrész tulajdonságai megváltoznának

Egy ősosztályból származó osztályt mindenhol fel kell tudni használni, ahol az ősosztályt várjuk

```
class Rectangle {
    private $topLeft;
    private $width;
    private $height;

    public function setHeight($height) {
        $this->height = $height;
    }
    public function getHeight() {
        return $this->height;
    }
    public function setWidth($width) {
        $this->width = $width;
    }
    public function getWidth() {
        return $this->width;
    }
}
```

```php
class Square extends Rectangle {
    public function setHeight($value) {
        $this->setSize($value);
    }
    public function setWidth($value) {
        $this->setSize($value);
    }
    private function setSize($value) {
        $this->width = $value;
        $this->height = $value;
    }
}
```

```php
class RectangleDisplayer {
    public function showLeftBorder(Rectangle $rectangle) {
        $rectangle->setWidth(2);
        $rectangle->setHeight(50);
        $this->display($rectangle);
    }
}
```

SOLID

INTERFACE SEGREGATION PRINCIPLE

```php
interface User {
    public function getId();
    public function getName();
    public function getAge();
    public function getAddress();
    public function getEmail();
    public function getPhone();
    public function getMothersMaidenName();
}


class EmailSender {
    public function sendEmail(User $user, $message) {
        ...
    }
}
```

```
class Community implements User {
    public function getId() { ... };
    public function getName() { ... };
    public function getAge() { ... };
    public function getAddress() { ... };
    public function getEmail() { ... };
    public function getPhone() { ... };
    public function getMothersMaidenName() {
        throw new WhatIsThisException();
    };
}

class EmailSender {
    public function sendEmail(User $user, $message) {
        ...
    }
}
```

```
interface Contactable {
    public function getEmail();
    public function getPhone();
}

interface Identity {
    public function getId();
    public function getName();
}

interface Addressable {
    public function getAddress();
}

interface Person {
    public function getMothersMaidenName();
}
```

```php
class Community implements Identity, Addressable, Contactable {
    public function getId() { ... };
    public function getName() { ... };
    public function getAddress() { ... };
    public function getEmail() { ... };
    public function getPhone() { ... };
}

class EmailSender {
    public function sendEmail(Contactable $contact, $message) {
        ...
    }
}
```

SOLID

DEPENDENCY INVERSION PRINCIPLE

- A magasabb szinten lévő programegységek nem függhetnek az alacsonyabb szintűektől, ehelyett mindkettőnek az absztrakciótól kell függenie.
- Az absztrakció nem függ a részletektől, a részletek függnek az absztrakciótól.

TIPIKUS PÉLDA: KÜLÖNBÖZŐ TÍPUSÚ ADATBÁZISOK KEZELÉSE

```
class UserController extends Controller {

    public function actionGetList() {
        $users = $this->query(
          'SELECT * FROM users ORDER BY created, desc;'
        );

        $view = new View('/userList');
        $view->render($users);
    }

}
```

ÜZLETI LOGIKA ▸ ABSZTRAKCIÓS RÉTEG ▸ ADATBÁZIS

```
class UserController extends Controller {

    public function actionGetList() {
        $users = UserRepository::getAll()
          ->orderBy('createdBy', 'desc')->execute();

        $view = new View('/userList');
        $view->render($users);
    }

}
```

ÜZLETI LOGIKA ▸ ABSZTRAKCIÓS RÉTEG ▸ ADATBÁZIS

```
class UserController extends Controller {

    public function actionGetList() {
        View::create('/userList')->render(
          UserRepository::create()->getListByCreation()
        );
    }

}
```

DEPENDENCY INJECTION

TARTSUK ÉSZBEN, HOGY MINDEN ABSZTRAKCIÓNAK ÁRA VAN!

DANKE!