Ktoś się może nawet oburzy, że o czym ja tu piszę. Przecież w PHP nie ma przeciążania znanego z C / Javy. Istnieje co prawda coś, co jest nazywane " przeciążaniem ", jednak działa na innej zasadzie. Ja jednak nie znalazłem innej nazwy. No może "statyczne przeciążanie konstruktora". Zaraz postaram się wyjaśnić, o co mi chodzi.

Na początek

- podstawowa wiedza o PHP5 (konstruktory, pola statyczne)
- serwer www (może być lokalny, np. WAMP)

Zwykła klasa

Oto kod zwykłej klasy, która obudowuje funkcję fopen :

{codecitation class='brush: php'}class File { private $sFilePath; private $hFileHandler; public function __construct($file_path) { $this->sFilePath = $file_path; } public function open($mode) { $this->hFileHandler = fopen($this->sFilePath, $mode); } public function write($content) { return fwrite($this->hFileHandler, $content); } } $oFile = new File('plik.txt'); $oFile->open('w+'); $oFile->write('test');{/codecitation}

Spokojnie. Jestem świadomy, że powyższy kod nie jest idealny. Zaraz go zmienimy. Jak widać, podstawowe problemy:

1. wywołanie konstruktora wcale nie otwiera pliku. Może należałoby dać tam 2-gi parametr, będący trybem, w jakim chcemy plik otworzyć.
2. trzeba pamiętać, żeby wywoływac najpierw File::open(), a potem File::write(). Inaczej czeka nas próba zapisania do nieotwartego pliku => błąd.

To jest zły kod.

ad 1: wstawienie drugiego parametru pozwoli po raz kolejny na błędy. Niech się komuś zdarzy pomyłka i zamiast 'w+' da 'e+'... To się może zdarzyć. Najprowdopodobniej dodatkowo zdarzy

się to w miejscu, gdzie nikt nie zauważy, bo rzadko tamten kod jest wywoływany. Ten bug ujawni się w wigilię, gdy będziesz zasiadał do kolacji, a klient właśnie będzie chciał uruchomić nową promocję...

Dodatkowym problemem jest to, że nie sprawdzam, czy plik w ogóle istnieje. Należy to jakoś załatać.

Propozycja rozwiązania

Być może lepiej jest mieć kilka konstruktorów. Czy nie byłoby dobrze, móc zrobić tak:

{codecitation class='brush: php'}

$oFile = new FileToWrite('plik.txt'); $oFile->write('test');

{/codecitation}

Możemy oczywiście zrobić nową klasę o takiej nazwie, która by dziedziczyła po File. Ja chciałbym jednak zaproponować coś innego.

"Przeciążanie" konstruktorów

{codecitation class='brush: php'}

class File { private $sFilePath; private $sMode; private $hFileHandler = null; const WRITE = 'w+'; // 1 const READ = 'r+'; const ADD = 'a+'; static function forceOpenToWrite($file) { // 2 self::createFileIfNotExists($file); return self::openToWrite($file); } static function forceOpenToAdd($file) { // 2 self::createFileIfNotExists($file); return self::openToAdd($file); } static public

function openToWrite($file) { // 3 return new self($file, self::WRITE); } static public function openToRead($file) { // 3 return new self($file, self::READ); } static public function openToAdd($file) { // 3 return new self($file, self::ADD); } static private function createFileIfNotExists($file_path) { // 4 if (!file_exists($file_path)) { $file = fopen($file_path, self::WRITE); fclose($file); } } private function __construct($file_path, $mode) { // 5 $this->sMode = strtolower($mode); $this->sFilePath = $file_path; if (!$this->fileExistsOrCanBeCreate($file_path)) { // 6 throw new Exception("Nie ma takiego pliku"); } $this->open($mode); } private function fileExistsOrCanBeCreate() { // 6 return file_exists($this->sFilePath) || self::WRITE === $this->sMode; } public function __destruct() { $this->close(); } public function read() { // 7 return file_get_contents($this->sFilePath); } public function write($content) { //7 return fwrite($this->hFileHandler, $content); } private function open($mode) { // 8 $this->hFileHandler = fopen($this->sFilePath, $mode); } private function close() { // 8 fclose($this->hFileHandler); $this->hFileHandler = null; } } $oFile = File::openToWrite('test1.txt'); $oFile->write('to jest test poprawny'); $oFile = File::openToAdd('test2.txt'); $oFile->write('to jest test poprawny'); $oFile = File::forceOpenToAdd('test2.txt'); $oFile->write('to jest test poprawny'); $oFile = File::openToRead('test3.txt'); echo $oFile->read();

{/codecitation}

Może krotko wyjaśnię ideę:

1. Używam stałych zamiast ciągów znaków. Zauważ w ilu miejscach stałe te są wykorzystane. Jeśli nagle będę chciał wykorzystać zamiast 'w+' samo 'w' albo 'wb+' zmiany ogranicze do jednego miejsca.
2. "Konstruktor" zwracający obiekt pliku do dopisywania lub pusty, Słowo `force' oznacz "jeśli podanego pliku nie ma - stwórz" -> patrz [4].
3. Metody zwracające obiekty odpowiednio do zapisu odczytu, dodawania. Nie wymuszają utworzenia pliku, jeśli on nie istnieje.
4. Metoda pomocnicza [stąd private] tworząca plik, jeśli nie istnieje.
5. Prywatny konstruktor. Skoro mamy w "statycznych konstruktorach" rozpatrzone wszystkie możliwe ścieżki to po co udostępniać publicznie konstruktor, który może zostać błędnie wywołany.
6. To jest jeden z moich fetyszy. Zamiast dziwnego i nic nie mówiącego warunku, wyrzuć go do osobnej metody i nazwij tak, abyś nie musiał komentować konstrukcji warunkowej.

7. Jedyne publiczne metody. W końcu ten obiekt ma jedynie zapisywać lub odczytywać dane do/z pliku.
8. Teraz już nie pozwalamy programiście decydować co i kiedy ma być otwarte zamknięte. On ma tylko robić to czego oczekujemy: zapisać lub odczytać dane. Tylko i aż tyle. Im mniej miejsc, gdzie może się pomylić tym lepiej.

Oczywiście powyższy kod wymagałby jeszcze poprawek. Przecież fopen wcale nie musi utworzyć pliku [za mało miejsca na dysku, brak uprawnień, setka innych powodów, których nie umiemy sobie wyobrazić]. Jednak tu chciałem pokazać pewną ideę, a nie cały gotowy system. Zapraszam do testowania.

Jeśli ktoś byłby zainteresowany, to udostępniam kolejne kroki jak przechodziłem od kodu nr 1 do ostatecznego:

- http://prezentacja-bs.googlecode.com/svn/trunk/inteligentne-obiekty/File_v1.class.php
- http://prezentacja-bs.googlecode.com/svn/trunk/inteligentne-obiekty/File_v2.class.php
- http://prezentacja-bs.googlecode.com/svn/trunk/inteligentne-obiekty/File_v3.class.php
- http://prezentacja-bs.googlecode.com/svn/trunk/inteligentne-obiekty/File_v4.class.php