

Programowanie obiektowe w PHP

v 1.2

Plan

- Dziedziczenie
- Zaawansowana obiektowość

Dziedziczenie

Dziedziczenie

Dziedziczenie to sposób, w jaki jedna klasa może przejmować funkcjonalność innej klasy.

Mówimy w takim przypadku, że klasa A dziedziczy po klasie B.

W przypadku dziedziczenia mówimy też często o relacji rodzic – dziecko.

W przypadku PHP rodzic może mieć wiele dzieci, ale dziecko może mieć tylko jednego rodzica.

Dziedziczenie – przykład

Bez dziedziczenia

```
class Book {  
    private $name = "Rok 1984";  
    private $price = 12.50;  
    private $author = "George Orwell";  
  
    ...  
}  
  
class Ebook {  
    private $name = "Rok 1984";  
    private $price = 12.50;  
    private $author = "George Orwell";  
    private $sizeInMB = 12;  
  
    ...  
}
```

Z dziedziczeniem

```
class Book {  
    private $name = "Rok 1984";  
    private $price = 12.50;  
    private $author = "George Orwell";  
  
    ...  
}  
  
class Ebook extends Book {  
    private $sizeInMB = 12;  
  
    ...  
}
```

Atrybut dostępu – protected

Protected – atrybut jest widoczny tylko z wnętrza danej klasy i klas dziedziczących.

Atrybut dostępu – protected

Kod

```
class Book {  
    private $name = "Rok 1984";  
    private $price = 12.50;  
    private $author = "George Orwell";  
  
    public function printBookInfo() {  
        echo($this->name. "<br>");  
        echo($this->price. "<br>");  
        echo($this->author. "<br>");  
    }  
}
```

Kod

```
class Ebook extends Book {  
    private $sizeInMB = 12;  
  
    public function printEBookInfo() {  
        echo($this->name. "<br>");  
        echo($this->price. "<br>");  
        echo($this->author. "<br>");  
        echo($this->sizeInMB. "<br>");  
    }  
}
```

Atrybut dostępu – protected

Kod – ciąg dalszy

```
$ebook1 = new Ebook();  
echo("Metoda printBookInfo <br>");  
$ebook1->printBookInfo();  
echo("Metoda printEBookInfo <br>");  
$ebook1->printEBookInfo();
```

Wynik

```
Metoda printBookInfo:  
Rok 1984  
12.5  
George Orwell  
  
Metoda printEBookInfo:  
Notice: Undefined property: Ebook::$name  
in index.php on line 30  
Notice: Undefined property: Ebook::$price  
in index.php on line 31  
Notice: Undefined property: Ebook::$author  
in index.php on line 32  
12
```


Atrybut dostępu – protected

Kod

```
class Book {  
    protected $name = "Rok 1984";  
    protected $price = 12.50;  
    protected $author = "George Orwell";  
  
    public function printBookInfo() {  
        echo($this->name. "<br>");  
        echo($this->price. "<br>");  
        echo($this->author. "<br>");  
    }  
}
```

Kod

```
class Ebook extends Book {  
    private $sizeInMB = 12;  
  
    public function printEBookInfo() {  
        echo($this->name. "<br>");  
        echo($this->price. "<br>");  
        echo($this->author. "<br>");  
        echo($this->sizeInMB. "<br>");  
    }  
}
```

Atrybut dostępu – protected

Kod – ciąg dalszy

```
$ebook1 = new Ebook();  
echo("Metoda printBookInfo <br>");  
$ebook1->printBookInfo();  
echo("Metoda printEBookInfo <br>");  
$ebook1->printEBookInfo();
```

Wynik

Metoda printBookInfo
Rok 1984
12.5
George Orwell

Metoda printEBookInfo
Rok 1984
12.5
George Orwell
12

Konstruktor i destruktor przy dziedziczeniu

- Podczas tworzenia obiektu silnik PHP użyje tylko konstruktora (lub destruktora) danej klasy.
- Jeżeli dana klasa nie ma konstruktora (lub destruktora), automatycznie dziedziczy go z klasy rodzica.
- Konstruktory (lub destruktory) klas nadrzędnych nie zostaną użyte (chyba że jasno wymusimy ich użycie).

Konstruktor i destruktor przy dziedziczeniu

Kod

```
class Book {
    private $name;
    private $price;
    private $author;

    public function __construct() {
        echo("Tworzę nową książkę<br>");
        $this->name = "Rok 1984";
        $this->price = 12.50;
        $this->author = "George Orwell";
    }

    public function __destruct() {
        echo("Niszczę książkę<br>");
    }
}
```

Kod

```
class Ebook extends Book {
    private $sizeInMB;

    public function __construct() {
        echo("Tworzę nowego ebooka<br>");
        $this->sizeInMB = 12;
    }

    public function __destruct(){
        echo("Niszczę ebooka<br>");
    }
}
```

Konstruktor i destruktor przy dziedziczeniu

Kod – ciąg dalszy

```
$ebook1 = new Ebook();
```

```
var_dump($ebook1);
```

Wynik

Tworzę nowego ebooka

```
object(Ebook)#1 (4) {  
    ["sizeInMB":"Ebook":private]=> int(12)  
    ["name":"Book":private]=> NULL  
    ["price":"Book":private]=> NULL  
    ["author":"Book":private]=> NULL  
}
```

Niszczę ebooka

Konstruktor i destruktor przy dziedziczeniu

Jawnie wymuszamy użycie konstruktora (i destruktor) klasy nadrzędnej przez użycie konstrukcji:

```
public function __construct(){  
    parent::__construct();  
    ...  
}
```

```
public function __destruct(){  
    parent::__destruct();  
    ...  
}
```

- Powinna to być zawsze pierwsza rzecz, jaką wywołujemy w konstruktorze klasy podrzędnej. Dla destruktorów powinna to być ostatnia rzecz.
- Wiele błędów polega na bazowaniu na niezainicjalizowanych zmiennych klasy nadrzędnej.

Konstruktor i destruktor przy dziedziczeniu

Kod

```
class Book {  
    private $name;  
    private $price;  
    private $author;  
  
    public function __construct() {  
        echo("Tworzę nową książkę<br>");  
        $this->name = "Rok 1984";  
        $this->price = 12.50;  
        $this->author = "George Orwell";  
    }  
  
    public function __destruct() {  
        echo("Niszczę książkę<br>");  
    }  
}
```

Kod

```
class Ebook extends Book {  
    private $sizeInMB;  
  
    public function __construct() {  
        parent::__construct();  
        echo("Tworzę nowego ebooka<br>");  
        $this->sizeInMB = 12;  
    }  
  
    public function __destruct() {  
        echo("Niszczę ebooka<br>");  
        parent::__destruct();  
    }  
}
```


Konstruktor i destruktor przy dziedziczeniu

Kod – ciąg dalszy

```
$ebook1 = new Ebook();  
  
var_dump($ebook1);
```

Wynik

Tworzę nową książkę
Tworzę nowego ebooka

```
object(Ebook)#1 (4) {  
    ["sizeInMB":"Ebook":private]=> int(12)  
    ["name":"Book":private]=> string(8) "Rok  
    1984"  
    ["price":"Book":private]=> float(12.5)  
    ["author":"Book":private]=> string(13)  
    "George Orwell"  
}
```

Niszczę ebooka
Niszczę książkę

Nadpisywanie metod (method overwriting)

- W PHP mamy możliwość nadpisywania metod (**method overwriting**).
- Służy to stworzeniu nowej funkcjonalności dla klasy podrzędnej, która ma być wywołana przez funkcję o tej samej nazwie co w klasie nadrzędnej.
- Jeżeli chcemy wykorzystać funkcję klasy nadrzędnej, możemy się do niej odwołać przez słowo kluczowe **parent::**

Nadpisywanie metod (method overwriting)

Kod

```
class Book {  
    private $name;  
    private $price;  
    private $author;  
  
    public function __construct() {  
        $this->name = "Rok 1984";  
        $this->price = 12.50;  
        $this->author = "George Orwell";  
    }  
  
    public function printBookInfo() {  
        echo($this->name."<br>");  
        echo($this->price."<br>");  
        echo($this->author."<br>");  
    }  
}
```

Kod

```
class Ebook extends Book {  
    private $sizeInMB;  
  
    public function __construct() {  
        parent::__construct();  
        echo("Tworzę nowego ebooka<br>");  
        $this->sizeInMB = 12;  
    }  
  
    public function printBookInfo() {  
        parent::PrintBookInfo();  
        echo($this->sizeInMB."<br>");  
    }  
}
```

Nadpisywanie metod (method overwriting)

Kod – ciąg dalszy

```
$ebook1 = new Ebook();
```

```
$ebook1->printBookInfo();
```

Wynik

Rok 1984

12.5

George Orwell

12

Czas na zadania

- Przeróbcie ćwiczenia z części B.
Pierwsze ćwiczenie zróbcie z wykładowcą.

Zaawansowana obiektoowość

Słowo kluczowe static

- Wszystkie przykłady, które pokazywaliśmy do tej pory, działały tylko przy użyciu obiektów.
- Jest jednak możliwość wywoływania pewnych funkcji przy odwołaniu się do klasy, a nie do danego obiektu.
- Taką funkcjonalność daje nam słowo kluczowe **static**.

Słowo kluczowe static – atrybuty

Jeżeli dodamy słowo kluczowe **static** do atrybutu, to stworzymy zmienną, która jest współdzielona przez wszystkie obiekty danej klasy. Przyjęło się, że to słowo kluczowe umieszczamy przed modyfikatorem dostępu.

Zawsze nastawiaj wartość domyślną takiej zmiennej.

```
class Book {  
    static public $aNum = 0;  
    ...  
}
```

Słowo kluczowe static – atrybuty

Z wnętrza klasy możemy się odnieść do takiej zmiennej przez konstrukcję:

self::\$nazwa_zmiennej

Z innych miejsc w kodzie przez:

Nazwa_Klasy::\$nazwa_zmiennej

```
class Book {  
    static public $aNum = 0;  
    public function IncreaseStaticNum(){  
        self::$aNum++;  
    }  
}
```


Słowo kluczowe static – atrybuty

Kod

```
class Book {  
    static public $aNum = 0;  
}
```

```
echo(Book::$aNum);
```

```
Book::$aNum = 12;
```

```
echo(Book::$aNum);
```

Wynik

0
12

Słowo kluczowe static – metody

- Jeżeli dodamy słowo kluczowe **static** do metody, to będzie ona działała dla klasy. Przyjęło się, że to słowo kluczowe umieszczamy przed modyfikatorem dostępu.
- Metoda taka nie może się odnosić do żadnej zmiennej, która nie jest statyczna.

```
class Book {  
    static public $aNum = 0;  
    static public function sayHello() {  
        self::$aNum++;  
        print("hello (" . self::$aNum . ")<br>");  
    }  
    ...  
}
```

Słowo kluczowe const

- Czasami chcemy wprowadzić atrybut, który nie może być zmieniony podczas działania naszej aplikacji.
 - W klasach możemy taki atrybut wprowadzić, używając słowa kluczowego **const**.
 - Atrybut taki jest atrybutem należącym do klasy, a nie do poszczególnych obiektów.
- Atrybuty **const** mogą przyjmować tylko wartości podstawowych typów danych (**int**, **bool**, **float**, ...).
 - Atrybutów **const** nie poprzedzamy znakiem \$.
 - Nazewnictwo takich atrybutów powinno spełniać następujące zasady:
 - używamy tylko dużych liter oraz numerów,
 - słowa oddzielamy znakiem podkreślenia: _.

Słowo kluczowe const

```
class Book {  
    const HARD_COVER = 0;  
    const SOFT_COVER = 1;  
    const NO_COVER = 2;  
    ...  
}
```

Klonowanie obiektów

Klonowanie obiektów wiąże się z wieloma problemami:

- w wersji PHP 4 jest inne niż w PHP 5 (co spowodowało błędne działanie wielu skryptów),
- przy bardziej skomplikowanej strukturze obiektu może wymagać dodatkowej pracy niż tylko przekopiowanie danych.

Klonowanie obiektów

Klonowanie obiektów odbywa się przez użycie słowa kluczowego **clone**.

```
$book1 = new Book();
```

```
$book2 = $book1;
```

\$book1 oraz \$book2 odnoszą się do tego samego obiektu.

```
$book3 = clone $book1;
```

\$book3 odnosi się do nowego obiektu który jest klonem \$book1

Wynik

```
object(Book)#1 (1) { ["aNum"]=> int(0) } //book1
```

```
object(Book)#1 (1) { ["aNum"]=> int(0) } //book2
```

```
object(Book)#2 (1) { ["aNum"]=> int(0) } //book3
```

Klonowanie obiektów

Co dokładnie się dzieje podczas klonowania obiektów?

- tworzony jest nowy obiekt (nie jest wywoływany konstruktor!!!),
- wszystkie atrybuty starego obiektu są kopiowane płytko (**shallow copy**) do nowego obiektu (oznacza to, że użyty jest operator =),
- wywoływana jest magiczna metoda **__clone()**.

Magiczna metoda

Magiczna metoda **__clone** powinna zawsze implementować następujące kroki:

- czyścić informacje, które mają być unikalne dla obiektu (np. id obiektu),
- wywoływać **__clone** na wszystkich obiektach, które nie mają być współdzielone między obiektem klonowanym a jego klonem.

Głębokie klonowanie (deep copy)

Kod

```
class Bookmark{
    public $pageNo;
    public function __construct($newPageNo) {
        $this->pageNo = $newPageNo;
    }
}

class Book{
    private $name;
    public $myBookMark;
    public function __construct($newName) {
        $this->name = $newName;
        $this->myBookMark = null;
    }
}
```

Kod

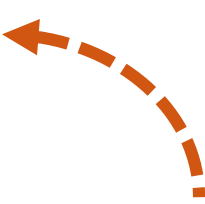
```
public function printBookInfo() {
    echo("Nazwa książki to: ".$this
        ->name."<br>");
    if($this->myBookMark != null) {
        echo("Zakładka jest na stronie: ".
            $this->myBookMark
            ->pageNo."<br>");
    }
}
```

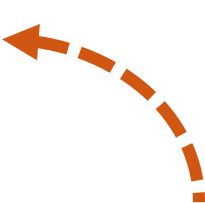

Głębokie klonowanie (deep copy)

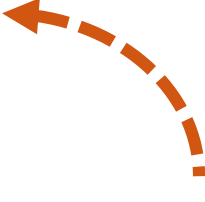
Kod – ciąg dalszy

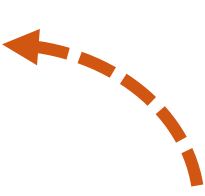
```
$book1 = new Book("paragraf 22");  
$book1->myBookMark = new Bookmark(15);  
$book2 = clone $book1;  
$book1->printBookInfo();  
$book2->printBookInfo();  
$book1->myBookMark->pageNo = 70;  
$book1->printBookInfo();  
$book2->printBookInfo();
```

Wynik

Nazwa książki to: paragraf 22
Zakładka jest na stronie: 15  \$book1

Nazwa książki to: paragraf 22
Zakładka jest na stronie: 15  \$book2

Nazwa książki to: paragraf 22
Zakładka jest na stronie: 70  \$book1

Nazwa książki to: paragraf 22
Zakładka jest na stronie: 70  \$book2

Głębokie klonowanie (deep copy)

Kod

```
class Book{
    private $name;
    public $myBookMark;
    public function __construct($newName) {
        $this->name = $newName;
        $this->myBookMark = null;
    }
    public function printBookInfo() {
        echo("Nazwa książki to: ".$this->name."<br>");
        if($this->myBookMark != null) {
            echo("Zakładka jest na stronie: ".
                $this->myBookMark->pageNo."<br>");
        }
    }
}
```

Kod

```
public function __clone() {
    if($this->myBookMark != null) {
        $this->myBookMark = clone $this->myBookMark;
    }
}
```

Głębokie klonowanie (deep copy)

Kod – ciąg dalszy

```
$book1 = new Book("paragraf 22");  
$book1->myBookMark = new Bookmark(15);  
$book2 = clone $book1;  
$book1->printBookInfo();  
$book2->printBookInfo();  
$book1->myBookMark->pageNo = 70;  
$book1->printBookInfo();  
$book2->printBookInfo();
```

Nazwa książki to: paragraf 22
Zakładka jest na stronie: 15



\$book1

Nazwa książki to: paragraf 22
Zakładka jest na stronie: 15



\$book2

Nazwa książki to: paragraf 22
Zakładka jest na stronie: 70



\$book1

Nazwa książki to: paragraf 22
Zakładka jest na stronie: 15



\$book2

Czas na zadania

- Przeróbcie ćwiczenia z części C
Pierwsze ćwiczenie zróbcie z wykładowcą.