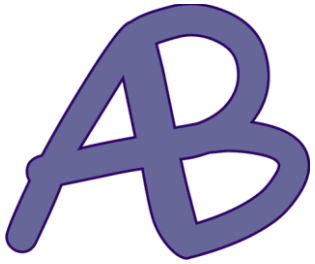


VI. PHP

- Klasy

dr Artur Bartoszewski
UTH Radom

Klasy

Klasy

Definicja klasy w PHP

```php
<?php
class Osoba
{
  public $imie;
  public $nazwisko;
  public $wiek;
}
?>
```

Tworzenie obiektu

```php
<?php
$ktos = new Osoba();
$ktos->imie = "Jan";
$ktos->nazwisko = "Kowalski";
$ktos->wiek = 40;
?>
```

Domyślne wartości pól

```php
<?php
class Osoba
{
    public $imie = 'NN';
    public $nazwisko = 'NN';
    public $wiek = 0;
}
?>
```

Metody klas

```
class Osoba
{
    private $imie = 'NN';
    private $nazwisko = 'NN';
    private $wiek = 0;
    public function setImie($kto)
    {
        $this->imie = $kto;
    }
    public function getImie()
    {
        return $this->imie;
    }
}
```

Aby dostać się do pola w konkretnym obiekcie, na którym wykonuje się metodę, należy skorzystać ze zmiennej $this, która reprezentuje obiekt tej klasy.

Domyślnie właściwość opakowujemy dwoma metodami:

Get – służy do pobrania wartości.

Set – służy do nadania wartości.

Prawa dostępu

W każda metoda klasy lub jej zmienna może być zadeklarowana za pomocą jednego z trzech słów kluczowych:

1. public (publiczna) – zmienna lub metoda jest widoczna z całego skryptu,
2. protected (chroniona) – zmienna lub metoda jest widoczna tylko z obiektu, w którym się znajduje, bądź z jego obiektu podrzędnego,
3. private (prywatna) – zmienna lub metoda jest widoczna tylko z obiektu, w którym się znajduje.

Prawa dostępu

W każda metoda klasy lub jej zmienna może być zadeklarowana za pomocą jednego z trzech słów kluczowych:

1. public (publiczna) – zmienna lub metoda jest widoczna z całego skryptu,
2. protected (chroniona) – zmienna lub metoda jest widoczna tylko z obiektu, w którym się znajduje, bądź z jego obiektu podrzędnego,
3. private (prywatna) – zmienna lub metoda jest widoczna tylko z obiektu, w którym się znajduje.

Dziedziczenie

```php
class Osoba
{
    private $imie = 'NN';
    private $nazwisko = 'NN';
    private $wiek = 0;
    public function setImie($kto)
    {
        $this->imie = $kto;
    }
    public function getImie()
    {
        return $this->imie;
    }
}
```

```php
class Pracownik extends Osoba
{
    private $stanowisko;
    public function setStanowisko($kto)
    {
        $this->stanowisko = $kto;
    }
    public function getStanowisko()
    {
        return $this->stanowisko;
    }
}
```

final - klasa finalna – zablokowanie dziedziczenia

```php
class Osoba
{
    private $imie = 'NN';
    private $nazwisko = 'NN';
    private $wiek = 0;
    final public function setImie($kto)
    {
        $this->imie = $kto;
    }
}
```

Używając słowa kluczowego **final** można zabronić nadpisywania metod.
Można też użyć modyfikatora **final** dla całej klasy, wtedy żadna inna klasa nie będzie mogła po niej dziedziczyć.

```php
final class Osoba
{
    private $imie = 'NN';
    private $nazwisko = 'NN';
    private $wiek = 0;
}
```

abstarct – metoda i klasa abstrakcyjna – przeznaczona tylko do dziedziczenia dziedziczenia

```php
abstract class Osoba
{
    private $imie;
    abstract public function setImie($kto);
    abstract public function getImie();
}

class pracownik extends Osoba
{
    public function setImie($kto){
        $this->imie = $kto;
    }
    public function getImie(){
        return $this->imie;
    }
}
```

Można zadeklarować metodę jako abstrakcyjną.

W takiej sytuacji jej deklaracja **nie może zawierać ciała** (kodu do wykonania).

Należy pamiętać, że aby tego dokonać, klasa również musi być zadeklarowana jako abstrakcyjna.

Dopiero w klasie rozszerzającej **tworzymy ciało metod abstrakcyjnych**.

Deklaracja musi być zgodna z późniejszymi implementacjami. Gdy więc określimy argumenty, ich typy czy typ zwracany w abstract, to metody w klasach dziedziczących muszą do tego pasować.

Interfejsy

- Klasy mogą dziedziczyć tylko z jednej klasy nadrzędnej. Prowadziło to czasami do nadmiernego
- komplikowania drzewa klas.
- Aby temu zapobiec wprowadzono interfejsy, które w rzeczywistości nie mają żadnych metod. Interfejs to tylko definicja metody i parametrów, które ona pobiera.
- Aby dołączyć interfejs, należyvsię posłużyć słowem implements.
- Do klasy można później zaimplementować dowolną liczbę interfejsów, pod warunkiem że w klasie zostaną zadeklarowane metody z interfejsów.
- Jeśli nie będzie zadeklarowana jakaś metoda wymagana przez zaimplementowany interfejs, wyświetli się fatal error.

Interfejsy

```
interface Adres {
    function podajAdres($adres);
}

interface Osoba {
    function setImie($kto);
    function getImie();
}
```

```
class Pracownik implements Osoba, Adres {
    private $imie;
    private $adres;
    function podajAdres($adres){
        $this->adres = $adres;
    }
    public function setImie($kto){
        $this->imie = $kto;
    }
    public function getImie(){
        return $this->imie;
    }
}
```

Pola i metody statyczne

- Metody i zmienne statyczne to takie, do których dostęp można uzyskać z zewnątrz (bez konieczności definiowania obiektu danej klasy).
- Elementy statyczne definiowane są za pomocą operatora static, a dostęp do nich musi być ustawiony na publiczny.
- Nie można też używać wyrażenia $this-> jako odniesienia do aktualnej klasy. Zamiast tego używajmy self::.
- Zmienne z metody statycznej nie mogą być pobierane za pomocą operatora ->. Zamiast tego używajmy operatora :: (podwójnego dwukropka).

Pola i metody statyczne

```
class liczbaPi {
    public static $wartosc = 3.141592;
    public static function podajPi()
    {
        return self::$wartosc;
    }
}

echo liczbaPi::$wartosc;
echo liczbaPi::podajPi();
```

Klasy cz. 2
„Magiczne metody”

Magiczne metody to metody, które nie są bezpośrednio używane przez programistę, lecz są automatycznie wywoływane przez interpreter w odpowiedzi na poszczególne zachowania danej klasy,

Tworzymy je jako metody klasy z zarezerwowanym prefiksem w postaci podwójnego podkreślenia „__”.

__construct() - konstruktor

```
class Test
{
    public $dana;
    public function __construct()
    {
        echo 'Zadziałał konstruktor<br>';
    }
}
```

Funkcja uruchamiana w momencie utworzenia obiektu, wykorzystujemy ją do przygotowania wszystkich potrzebnych danych dla klasy przed rozpoczęciem pracy.

__destruct() - destruktor

```
class Test
{
    public $dana;
    public function __destruct()
  {
        //akcja destruktora
        // np. zamykanie pliku
    }
}
```

Destruktor wywoływany jest na zakończenie pracy danego obiektu przez garbage collector, zwalniając zajęte przez klasę zasoby.
Możemy go wykorzystać np. do zamknięcia wcześniej otwartego pliku we wnętrzu klasy albo rozłączenie z bazą danych.

__set()

Metoda wykonuje się, gdy próbujemy przypisać wartość do pola, które nie istnieje lub nie ma do niego dostępu.
Przyjmuje dwa argumenty,

- nazwa pola, do którego próbowaliśmy wpisać dane
- wartość jaką próbowaliśmy przypisać.

```php
class Test
{
  public $dana;
  public function __set($name, $value)
  {
   echo "Nie można przypisać wartości: ".$value." do pola: ".$name;
  }
}

$t1 = new Test;
$t1 -> xxx = 20;
?>
```

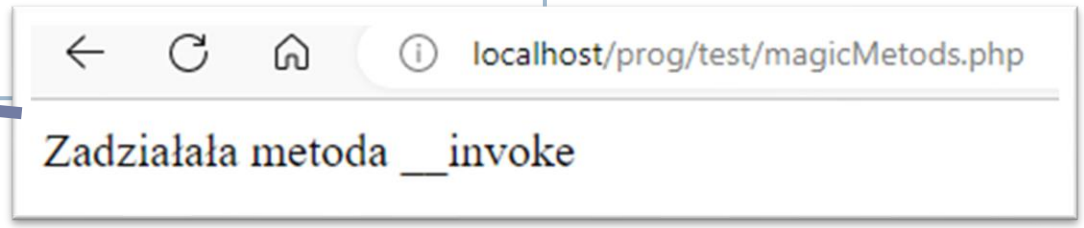
__invoke()

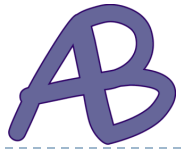
```php
class Test
{
  public $dana;

  public function __invoke()
  {
    echo "Zadziałała metoda __invoke";
  }
}

$t1 = new Test();
$t1();
```

Metoda __invoke() zostanie wykonana, gdy spróbujemy wywołać obiekt jako funkcję.





Literatura

W prezentacji użyto przykładów z książki:
- Żygłowicz Jerzy - PHP - Kompendium wiedzy, Helion

- https://www.php.net